MACH® Devices Applications Handbook

First Edition

Advanced
Micro
Devices

MACH® Devices

Applications Handbook

First Edition

A D V A N C E D   M I C R O   D E V I C E S

Special thanks to the people that made this book possible:

Kristin Ahrens, Ray Broemmelsiek, Ken Carey, Mary Carrig, Tami Conover, Gene Crossley, Günter Delfs, Christine DiCristo, Arlene Edwards, Dianne Hicks, Stephen Leis, Ulrich Lonsdorfer, Phil Madison, Kamru Miah, Bryon Moyer, Christoph Niessen, Uwe Nolte, Yvonne Quevedo, Stuart Tindall, Ken Wood, and Shawn Worsell.

Trademarks

TABLE OF CONTENTS

INTRODUCTION

When the MACH® Family was introduced in 1990, the intent was to address a series of needs that designers were expressing for higher-density logic. These included high-density, non-volatile, field-programmable devices that did not saddle the designer with burdensome costs and speed degradation. The MACH 1 and 2 Families have overwhelmingly succeeded in meeting these goals by providing electrically-erasable, high-density devices at extremely high speeds.

Since their inception, the MACH 1 and 2 device family members have been used in a wide variety of applications. These designs range from consolidating a number of smaller PAL® devices into a larger MACH device, to applications requiring very large scale integration.

The MACH Devices Applications Handbook, First Edition, contains a sampling of the variety of different designs that customers have incorporated into MACH devices. This collection of actual customer solutions illustrates the flexibility of these devices; from the 44-pin MACH110, to the 84-pin MACH230.

The majority of these designs were obtained from European customers that entered their solutions in a MACH Design Challenge that occurred in 1992. Some of the designs were originally written in German and were subsequently translated. Both text versions are included for authenticity. Since many designs originate in restaurants, a number of the drawings were enhanced to provide a clearer image of the author's intent. The Fitter Files were condensed in order to save trees. The Design File and Placement File for each design were not altered.

The Design Files were created on a number of different software packages. These include AMD's PALASM™ 4 Software, Data I/O's ABEL™ Software, and Logical Devices' CUPL™ Software. This illustrates the flexibility of designing with MACH devices by using software packages provided by our third party FusionPLD℠ Partners Program.

As these designs are actual customer submissions, they are intended as guidelines only.

New MACH device applications are created every day and it is therefore the intent of AMD to update this collection of designs on a regular basis. If you would like to submit your MACH device application for consideration, please send it to the attention of the PLD Applications Group at the following address:

Advanced Micro Devices
901 Thompson Place
M/S 1028
P.O. Box 3453
Sunnyvale, California
94088-3453

We hope that you will find this Handbook helpful for your future solutions with MACH devices.

Happy designing!

Bus Access Controller: MACH110

Application Note

by Ken Wood, Digital Imaging Systems, United Kingdom

Advanced Micro Devices

INTRODUCTION

This paper describes an application for the AMD MACH110 programmable logic device. The device is programmed to perform memory access control functions for a modular framestore unit within an image capture, processing, and display system.

As will be seen, the framestore unit employs two MACH110's and a PAL22V10 to achieve its full range of control functions, but for proprietary reasons only one of the MACH110's will be described.

DESIGN REQUIREMENT

The motivation behind the design was to provide a modular mechanism by which the framestore memory of an image processing system could be configured to the needs of the application, merely by fitting the appropriate number of memory modules to make up the full storage requirement. The modules were to be physically identical, with no jumper or DIL-switch settings, all configuration options being handled by initialization code in the control processor software.

In addition, the modules were to handle data eight bits wide, but be capable of acting in concert to handle video input data up to 16 bits wide and output data up to 32 bits wide.

A balance had to be struck between complexity in the system controller, complexity in the framestore modules, and number of signal connections routed between. Simple framestores require a complex control scheme, and a large number of signals to implement the functions. The speed of such a scheme is limited due to the number of buffers and the physical distance between the control center and the units it acts on. By the converse argument, building a lot of autonomy into the modules reduces the complexity of the system controller and I/O, but the hardware is duplicated for each framestore module.

Combined with these specific requirements were the usual general requirements for a modern electronic design: maximum performance for minimum cost. More particular was a need to implement the module to occupy as little space as possible, so as to be able to fit the greatest number within any specified enclosure.

DESIGN SOLUTION

To answer these various needs, an architecture was chosen where there is sufficient autonomy on the framestore modules to sequence the high-speed operations, but complexity is held down by letting the system controller handle less frequent and slower operations.

Having a number of memory modules in a linear address space typically requires some upper address bits, decoded to form an enable signal unique to each module. This is often achieved either by performing the decoding in the system and routing the enable signal to each slot at which a memory module may be fitted, or by routing the upper address bits to each module so that a module decodes its own address set up by jumpers or encoded in the slot position.

The latter arrangement was chosen, and each module determines its address in the system by a hard-wired code on the connector. This meets the requirement to avoid any jumper link settings, but introduces the possibility that the installed memory modules may not form a contiguous block of addressable memory. To combat this, and to enhance the flexibility of the memory system, a virtual addressing scheme was devised such that each framestore module can be programmed to respond to a particular address, and is not restricted to the address allocated in the system.

Each framestore module includes a register, which can be written to by the system at an IO location defined by the Physical Address of the module (taken from the hard-wired code on the connector). The code in the register defines the address which the framestore module responds to for memory accesses, and thus acts as an address mapping table, except that the hardware which performs the address mapping is distributed across the system instead of being concentrated in a central memory management unit.

System reset forces the virtual address register in each module into a state representing the physical address of the module. Thus, from reset, the memory on each framestore module can be accessed at its physical address, and the system can perform a scan to determine the population of modules. Following this, virtual addresses for each module can be allocated to collect all the framestores into a contiguous block.

A further level of sophistication is included for video accesses to the framestore array. Each module handles data eight bits wide, but to handle the possibility of video input up to 16 bits wide there has to be the capability to pair up modules to handle the double width. Alternative data paths are available from either the most significant

Publication# 17748 Rev. A Amendment/0
Issue Date: May 1993

or least significant bytes of a 16-bit video data bus, and a second register on each framestore module holds the address to which the module will respond for video accesses.

The video address register operates in a similar way to the virtual address register, but with the twist that if two modules are programmed with the same video address, one on the most significant and the other on the least significant data bus, they will share a common address and appear as a 16-bit framestore module. Furthermore, it may be convenient under some circumstances to map the memory array in one way for system access and in another for video access: separate virtual and video address registers allow for this possibility.

The MACH110 under discussion implements the virtual addressing scheme and generates strobes to the various data path buffers. A further MACH110 implements the high speed sequencing for video output from the VRAMs, and one half of a 22V10 PAL provides various signal decodes. Figure 1-1 shows the block diagram for the framestore.

Fitting the MACHs in PLCC-to-PGA sockets (surface mount was not an option), and including all the VRAMs (zip packages), bus buffers, decoupling capacitors and connectors, has allowed a module comprising two complete framestore units to be squeezed onto a single-Eurocard size board (100x160 mm). Considering the complexity of the control logic, this is a substantial improvement over other implementations using PALs and MSI: one might estimate at least four PALs and two MSI chips to replace the first MACH and something similar for the second, and doubled up for the two framestores on the module.

DESIGN OF THE MACH

Figure 1-2 shows a block diagram for the Bus Access Controller MACH. The address space in use for the current access is decoded from the Access Type bus signals, and the appropriate recognition address routed via the 3:1 multiplexer out to an external address comparator (this function is difficult to implement in sum-of-product programmable devices). The Physical Address input is gated through for access to the Virtual Address Register, the video address for video accesses, and the virtual address for control processor accesses.

The Virtual Address Register is clocked on the trailing edge of the incoming data strobe, but designed to hold its current contents unless the Access Type is decoded as either Special IO (to write a new value) or Reset. For Special IO the value on the data bus is written to the register. For reset, the value on the Physical Address input is entered into the register, so that the control processor can access the framestore at its physical address until a virtual address is assigned to it.

Two further bits in the Virtual Address Register allow for the framestore to be half-populated with VRAM: if the Size flag is set to half, the framestore only responds to an access if the next lower address bit matches the Offset bit. If the Size flag indicates full, the Offset bit is ignored. At reset, the Size flag is initialized to full, then the control processor can assess how much VRAM is actually fitted and set the Size flag to suit. Two half-populated framstore modules can then be combined using the virtual addressing scheme to appear as one fully-populated module.

The Video Address Register is also clocked by the data strobe, and arranged to be written during the appropriate access and hold its value at other times. Its bits go to zero at reset, so must be initialized by the control processor prior to any video operation. An extra bit in the register defines whether the framestore is connected to the most significant or least significant bytes of the data bus, allowing two framestore modules to be paired to make a 16-bit module.

The other logic blocks identified in Figure 1-2 control the data flow within the MACH, and produce control signals for external events based on decodes of some of the internal and external signal inputs. This includes controlling some of the operation of the second MACH. The function of the various signals, both input and output, is fully documented in the ABEL sourcefile for the MACH program.

PROGRAMMING THE MACH

The design, as originally conceived, had to be adapted to allow a fit within the resource constraints of the MACH110 device. When a modification was found that would fit, it occupied 35 out of 38 pins, and 27 out of 32 macro cells. The device utilization was 88%.

With such a dense design, it was inevitable that the switch matrix would be congested, but there were also difficulties ensuring only 22 inputs were required to each half of the MACH. Design economies had to be made which meant sacrificing some functionality, but also led to crisper design elsewhere in the logic. It served to illustrate the old adage, that there is more than one way to skin a cat! Some of the steps taken to engineer a fit are described below.

Reset: It was originally intended to have a simple reset input, acting on the asynchronous terminal of the registers in the MACH, but there was no way to accommodate it within the 22-input limit for each half of the MACH. A compromise was made so that a code on the access type signals indicates reset, as these signals are already distributed.

This measure did not save a pin, however, as a reset decoder had to be added to generate a reset signal available to the rest of the framestore logic.

Valid: As originally planned, an extra bit in the Virtual Address Register would indicate whether a virtual address had been written to the register since reset. If not, the multiplexer would send the physical address out to the comparator instead of routing data from the Virtual Address Register, when appropriate.

This scheme led to an excessive number of product terms for the multiplexer, and it was realized that by initializing the Virtual Address Register with the physical address, this extra decision becomes unnecessary. Additionally, allowing the multiplexer outputs to be zero during don't care conditions reduced them to only occupy one macrocell each. One macrocell and its attendant routing is also saved by losing the 'valid' bit.

Offset: The Video Address Register was to have an offset bit independent of the one in the Virtual Address Register, specifically for video accesses. This was abandoned as it led to too great a complexity in the data path control and VRAM block select equations. Making processor accesses always via the LS byte of the data bus (independent of the DATAMS bit in the Video Address Register) also had to be abandoned for similar reasons.

CONCLUSION

The MACH110 was critical to the success of this design. It encapsulates a self-contained block of logic in a manner which minimizes the number of signals that cross the boundary of that block. A second MACH110, with similar complexity of function, completes the suite for the framestore unit. A larger programmable device, with sufficient resources to fulfill both functions, would require almost double the number of IO pins, therefore have double the perimeter (assuming PLCC) and quadruple the board area.

A device able to replace all four MACHs on a framestore module (two units) would be even more short of IO. Devices with less resources would only implement part of the logic function, thereby requiring more of them and increasing the IO requirement due to the interconnect between them.

Yet speed is also high on the list of critical characteristics. The maximum pixel rate sustainable by the framestore is defined by the delay through the various stages of memory control, and this is greatly aided by absorbing so much logic into a single device, with defined delays from input to output. Implementing the design in standard logic would impose a series of gate delays on each path, while competing high-density PLD architectures have poorer specifications (or worse, cannot be predicted in advance).

In summary, the 44-pin MACH110-15 (and the future MACH110-12) represents the optimum IO count/logic complexity/board area/speed trade-off for this application.

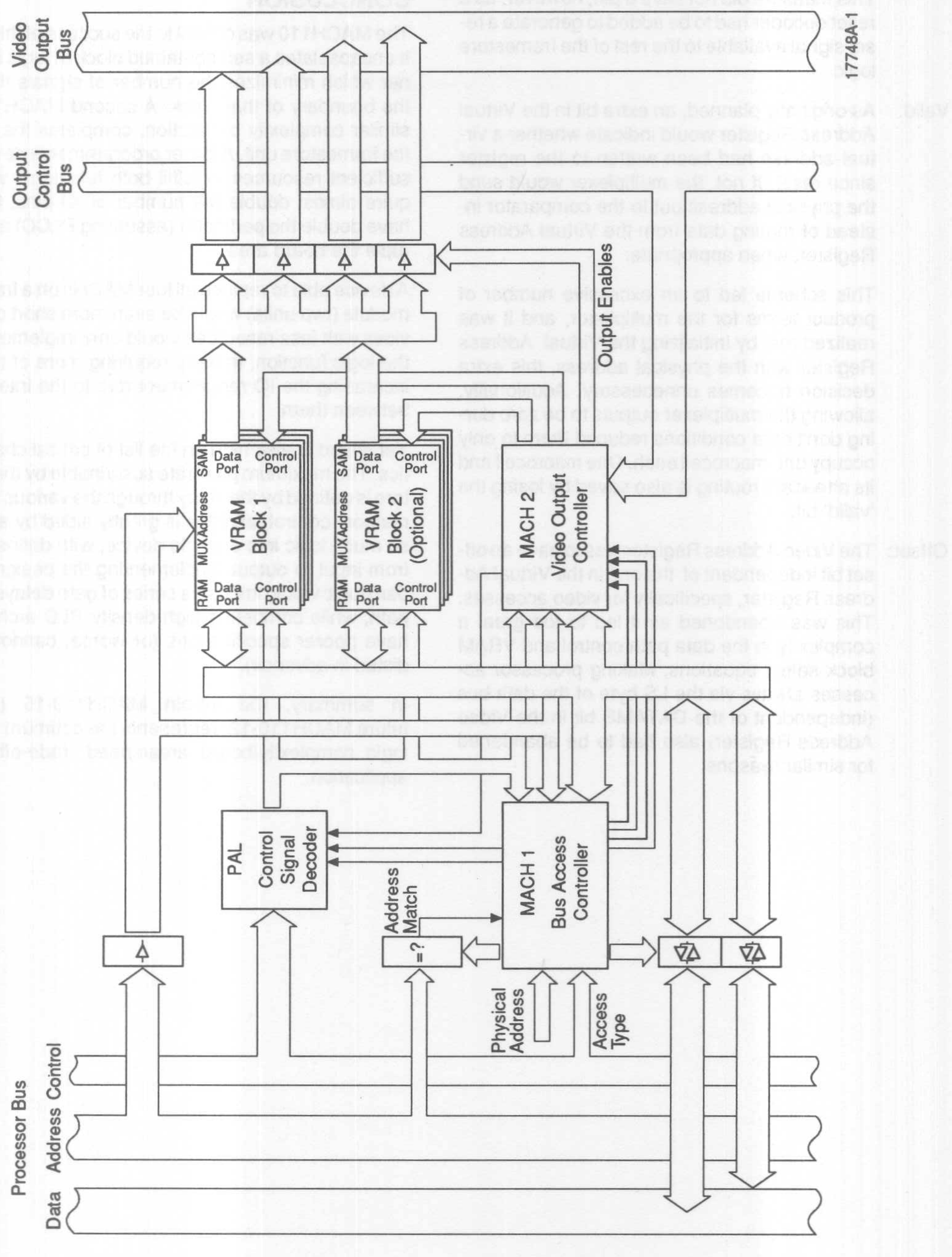


Figure 1-1. Block Diagram for the Framestore

17748A-2

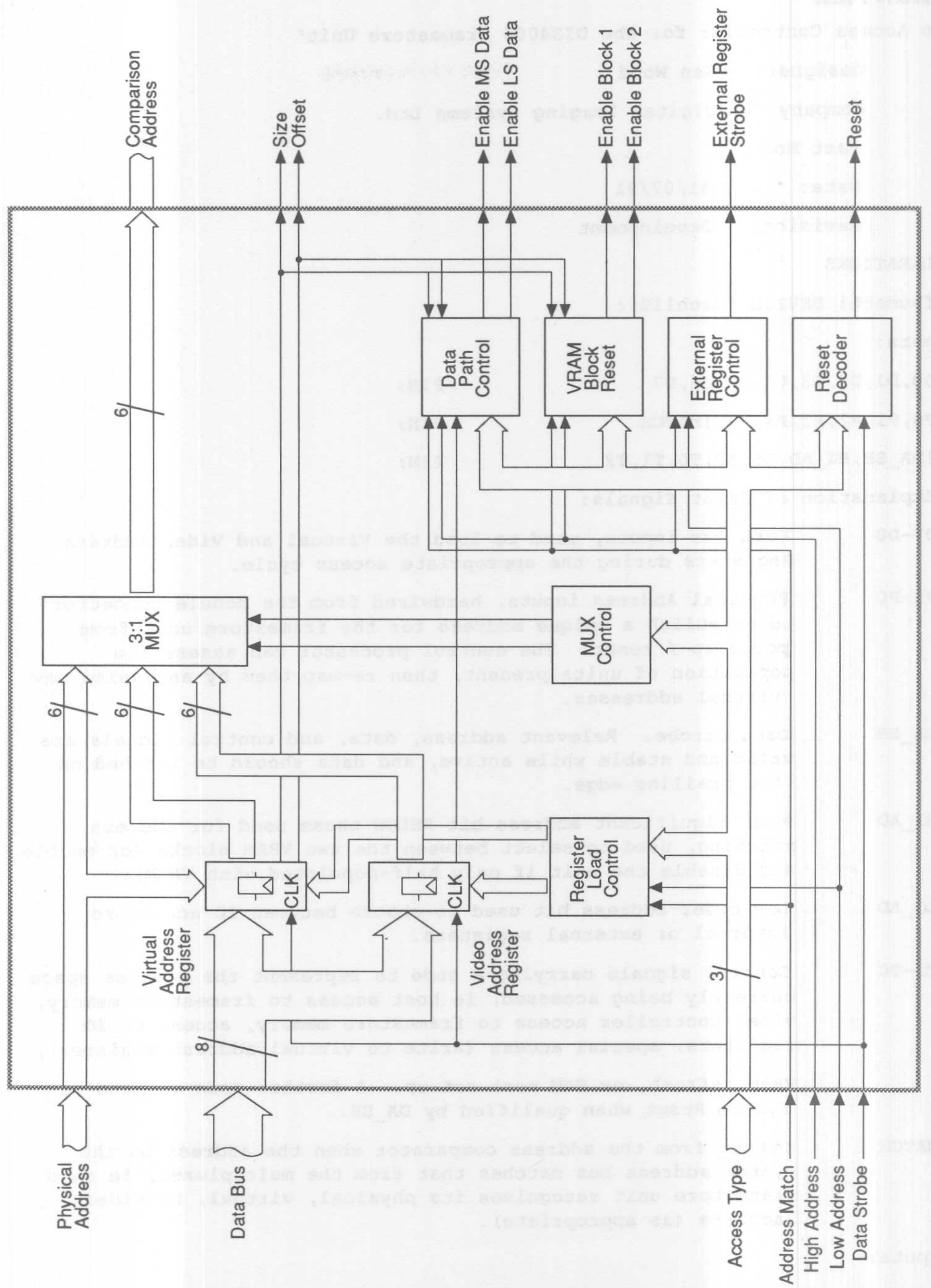


Figure 1-2. Block Diagram of the Bus Access Controller MACH110

DESIGN FILE

```
'Bus Access Controller for the DIS4000 Framestore Unit'
"        Designer:   Ken Wood
"        Company:    Digital Imaging Systems Ltd.
"        Part No:
"        Date:       31/07/91
"        Revision:   Development
DECLARATIONS
   fsumach1 DEVICE 'mach110';
"Inputs:
   D0,D1,D2,D3,D4,D5,D6,D7                    PIN;
   P0,P1,P2,P3,P4,P5,!MATCH                   PIN;
   !DA_SB,HI_AD,LO_AD,T0,T1,T2                PIN;
"  Explanation of Input Signals:
"  D7-D0       Data bus inputs, used to load the Virtual and Video Address
"              Registers during the appropriate access cycle.
"  P5-P0       Physical Address inputs, hardwired from the module connector
"              to establish a unique address for the framestore unit from
"              power up / reset.  The control processor can assess the
"              population of units present, then re-map them by assigning new
"               virtual addresses.
"  DA_SB       Data Strobe.  Relevant address, data, and control signals are
"              valid and stable while active, and data should be latched on
"               the trailing edge.
"  HI_AD       Most significant address bit BELOW those used for address
"              matching, used to select between the two VRAM blocks (or enable
"              and disable the unit if only half-populated with VRAM).
"  LO_AD       Low order address bit used to choose between IO access to
"              internal or external registers.
"  T2-T0       Control signals carrying a code to represent the address space
"              currently being accessed, ie host access to framestore memory,
"              video controller access to framestore memory, access to IO
"              registers, special access (write to virtual address register),
"              VRAM refresh, or SAM port set-up.  A further code is used as
"              System Reset when qualified by DA_SB.
"  MATCH       Output from the address comparator when the address on the
"              system address bus matches that from the multiplexer, ie this
"              framestore unit recognises its physical, virtual, or video
"               address (as appropriate).
"Outputs:
   C0,C1,C2,C3,C4,C5                          PIN ISTYPE 'com,buffer';
   SIZE,OFFSET                                PIN ISTYPE 'reg,buffer';
   !EBTOP,!EBBOT,!EBLS,!EBMS,!REGSB,!RESET PIN ISTYPE 'com,invert';
```

```
"  Explanation of Output Signals:

"  C5-C0          Comparison Address outputs from multiplexer to external
"                 address comparator.  The value on these signals is sourced
"                 from either the physical address inputs, or the virtual or
"                 video address registers according to the access type.

"  SIZE           Status flag indicating whether the framestore unit is fully
"                 populated with VRAM, or only half populated.  The flag co-
"                 resides in the Virtual Address Register, and is initialised to
"                 'full' at reset.  The control processor may then test memory
"                 and set to 'half' if appropriate.

"  OFFSET         Also co-residing in the Virtual Address Register, if SIZE=half
"                 this flag determines whether the unit is to act as the top
"                 half or bottom half of the addressable block, thus allowing
"                 another half-populated unit to be set to the same virtual
"                 address and fill in the missing half.

"  EBTOP,EBBOT    These signals enable access to the two VRAM blocks according
"                 to the access type, SIZE, OFFSET, and HI_AD.  If SIZE=half,
"                 EBBOT is never active.

"  EBLS,EBMS      These signals enable the data path buffers, either to the
"                 least or the most significant bytes on the processor data bus
"                 according to the DATAMS bit in the Video Address Register (see
"                 below).  This allows two framestore units to pair up and act
"                 as a single 16-bit framestore.

"  REGSB          A decoded data strobe for registers external to this device,
"                 it becomes active during IO accesses when LO_AD is high.

"  RESET          Ouput becoming active when Access Type = Reset and DA_SB is
"                 active, thus providing a reset signal local to the framestore
"                 unit.
"Internal nodes:

   VID0,VID1,VID2,VID3,VID4,VID5            NODE ISTYPE 'reg,buffer';

   VIR0,VIR1,VIR2,VIR3,VIR4,VIR5,DATAMS     NODE ISTYPE 'reg,buffer';

"  Explanation of Internal Nodes:

"

"  VIR5-VIR0      Virtual Address Register.  Loaded from P5-P0 at reset so that
"                 the virtual address equals the physical address until
"                 otherwise commanded, and updated from the data bus when access
"                 type = Special and address = physical address.  The virtual
"                 address is routed to the address comparator during host or IO
"                 accesses.  SIZE and OFFSET are also included in the register,
"                 but are not internal nodes as the signals are required
"                 externally.

"  VID5-VID0      Video Address Register.  Set to zero at reset, and loaded from
"                 the data bus when access type = IO, address = virtual address,
"                 and LO_AD is low.  The video address is routed to the address
"                 comparator during video accesses.

"  DATAMS         Co-resides in the Video Address Register, and can be set to
"                 make the framestore memory accessible on the most significant
"                 byte of the data bus.  This allows the framestore unit to pair
"                 up with another in a 16-bit configuration.
```

```
"MACH fitter block assignments:

AMDMACH property 'GROUP A C0 C5 SIZE OFFSET EBTOP EBBOT EBLS EBMS';

AMDMACH property 'GROUP A REGSB RESET VID0 VID5 DATAMS VIR0 VIR5';

AMDMACH property 'GROUP B C1 C2 C3 C4 VID1 VID2 VID3 VID4 VIR1';

AMDMACH property 'GROUP B VIR2 VIR3 VIR4';

"Sets:

   VIDREG   = [DATAMS,VID5..VID0];

   VIRREG   = [SIZE,VIR5..VIR0,OFFSET];

"Constants: Bus access types

   Host     = 0;

   Video    = 1;

   IO       = 2;

   Dummy    = 3;

   SAM      = 4;

   Refresh = 5;

   Special = 6;

   Rs       = 7;

"Constants: VRAM population control

   Half     = 1;

   Full     = 0;

"Miscellaneous identities to make equations more readable:

   HOST     = ( [T2..T0] == Host );

   VIDEO    = ( [T2..T0] == Video );

   REGIO    = ( [T2..T0] == IO );

   DUMMY    = ( [T2..T0] == Dummy );

   SPECIAL = ( [T2..T0] == Special );

   REFRESH = ( [T2..T0] == Refresh );

   SAMPORT = ( [T2..T0] == SAM );

   RS       = ( [T2..T0] == Rs );

   NORMACC = HOST # VIDEO # REGIO # DUMMY;

   FULL     = ( SIZE.fb == Full );

   HALF     = ( SIZE.fb == Half );

   SELTOP   = FULL & HI_AD # HALF & ( OFFSET.fb == HI_AD );

   SELBOT   = FULL & !HI_AD;

   ANYSEL   = SELTOP # SELBOT;
```

```
"Define extra identities to ease test vector generation:
   C,X      = .C.,.X.;
   TYPE     = [T2..T0];
   DATA     = [D6..D1];
   PHYS     = [P5..P0];
   COMP     = [C5..C0];
EQUATIONS
"Virtual Address Register:
   VIRREG.clk = !DA_SB;
   VIRREG.ar  = 0;
   VIRREG.ap  = 0;
   VIRREG.t   =   SPECIAL & MATCH & ( VIRREG.fb $ [D7..D0] )
                # RS & ( VIRREG.fb $ [Full,P5..P0,0] );
"Video Address Register:
   VIDREG.clk = !DA_SB;
   VIDREG.ar  = 0;
   VIDREG.ap  = 0;
   VIDREG.t   =   REGIO & MATCH & !LO_AD & ( VIDREG.fb $ [D7..D1] )
                # RS & ( VIDREG.fb $ [0,0,0,0,0,0,0] );
"Comparison Address Multiplexer:
   [C5..C0]   =   ( HOST # REGIO ) & [VIR5..VIR0].fb
                # VIDEO & [VID5..VID0].fb
                # SPECIAL & [P5..P0];
"Data Path Control:
   EBLS       = MATCH & ( SPECIAL
                        # ANYSEL & !DATAMS.fb & NORMACC
                        );
   EBMS       = MATCH & ANYSEL & DATAMS.fb & NORMACC;
"External Register Control:
   REGSB      = MATCH & LO_AD & REGIO & DA_SB;
"Reset Decoder:
   RESET      = RS & DA_SB;
"VRAM Block Select:
   EBTOP      =   REFRESH
                # SAMPORT
                # MATCH & SELTOP & ( HOST # VIDEO );
```

```
  EBBOT       =   REFRESH
                  # SAMPORT
                  # MATCH & SELBOT & ( HOST # VIDEO );
"  VERIFY DEVICE FUNCTION
"  **********************
TEST_VECTORS 'Check each comparison address output for proper function:'
( [ DA_SB, TYPE, MATCH, LO_AD, DATA, PHYS ] -> [ COMP ] )
@IRP X (0,63)  "Perform checks twice substituting 0's then 1's for X's.
{
" Check COMP tracks PHYS for access TYPE = Special:
  [   0,  Special, ?X,    ?X,     ?X,     9  ] -> [   9  ];
  [   0,  Special, ?X,    ?X,     ?X,    18  ] -> [  18  ];
  [   0,  Special, ?X,    ?X,     ?X,    36  ] -> [  36  ];
" Use reset to clock PHYS into VIRREG and clear VIDREG, and check (assuming
" correct codes are multiplexed out to COMP for various access types):
  [   C,     Rs,   ?X,    ?X,     ?X,     9  ] -> [   0  ];
  [   0,   Host,   ?X,    ?X,     ?X,    ?X  ] -> [   9  ];
  [   0,   Video,  ?X,    ?X,     ?X,    ?X  ] -> [   0  ];
  [   C,     Rs,   ?X,    ?X,     ?X,    18  ] -> [   0  ];
  [   0,   Host,   ?X,    ?X,     ?X,    ?X  ] -> [  18  ];
  [   0,   Video,  ?X,    ?X,     ?X,    ?X  ] -> [   0  ];
  [   C,     Rs,   ?X,    ?X,     ?X,    36  ] -> [   0  ];
  [   0,   Host,   ?X,    ?X,     ?X,    ?X  ] -> [  36  ];
  [   0,   Video,  ?X,    ?X,     ?X,    ?X  ] -> [   0  ];
" Test direct writes to VIRREG:
  [   C,  Special,  1,    ?X,      9,    ?X  ] -> [  ?X  ];
  [   0,   Host,   ?X,    ?X,     ?X,    ?X  ] -> [   9  ];
  [   0,   Video,  ?X,    ?X,     ?X,    ?X  ] -> [   0  ];
  [   C,  Special,  1,    ?X,     18,    ?X  ] -> [  ?X  ];
  [   0,   Host,   ?X,    ?X,     ?X,    ?X  ] -> [  18  ];
  [   0,   Video,  ?X,    ?X,     ?X,    ?X  ] -> [   0  ];
  [   C,  Special,  1,    ?X,     36,    ?X  ] -> [  ?X  ];
  [   0,   Host,   ?X,    ?X,     ?X,    ?X  ] -> [  36  ];
  [   0,   Video,  ?X,    ?X,     ?X,    ?X  ] -> [   0  ];
```

```
" Test direct writes to VIDREG:
 [   C,      IO,       1,      0,      54,    ?X  ] -> [  36  ];
 [   0,    Video,     ?X,     ?X,     ?X,     ?X  ] -> [  54  ];
 [   0,     Host,     ?X,     ?X,     ?X,     ?X  ] -> [  36  ];
 [   C,      IO,       1,      0,      45,    ?X  ] -> [  36  ];
 [   0,    Video,     ?X,     ?X,     ?X,     ?X  ] -> [  45  ];
 [   0,     Host,     ?X,     ?X,     ?X,     ?X  ] -> [  36  ];
 [   C,      IO,       1,      0,      27,    ?X  ] -> [  36  ];
 [   0,    Video,     ?X,     ?X,     ?X,     ?X  ] -> [  27  ];
 [   0,     Host,     ?X,     ?X,     ?X,     ?X  ] -> [  36  ];
" Ensure no cross-coupling between the registers:
 [   C,      Rs,      ?X,     ?X,     ?X,     63  ] -> [   0  ];
 [   C,      IO,       1,      0,      63,    ?X  ] -> [  63  ];
 [   C,  Special,      1,     ?X,      0,     ?X  ] -> [  ?X  ];
 [   0,    Video,     ?X,     ?X,     ?X,     ?X  ] -> [  63  ];
 [   C,      IO,       1,      0,      0,     ?X  ] -> [   0  ];
 [   C,  Special,      1,     ?X,      63,    ?X  ] -> [  ?X  ];
 [   0,    Video,     ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   C,  Special,      1,     ?X,      0,     ?X  ] -> [  ?X  ];
 [   C,      IO,       1,      0,      63,    ?X  ] -> [   0  ];
 [   0,     Host,     ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   C,  Special,      1,     ?X,      63,    ?X  ] -> [  ?X  ];
 [   C,      IO,       1,      0,      0,     ?X  ] -> [  63  ];
 [   0,     Host,     ?X,     ?X,     ?X,     ?X  ] -> [  63  ];
" Check COMP outputs for access type = Rs, Refresh, SAM is zero regardless
" of PHYS or register contents:
 [   0,      Rs,      ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   0,  Refresh,     ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   0,      SAM,     ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   C,  Special,      1,     ?X,      0,     ?X  ] -> [  ?X  ];
 [   C,      IO,       1,      0,      0,     ?X  ] -> [   0  ];
 [   0,      Rs,      ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   0,  Refresh,     ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
 [   0,      SAM,     ?X,     ?X,     ?X,     ?X  ] -> [   0  ];
```

```
" Check register contents cannot be disturbed by any transaction other than
" their proper write access:
  [   C,      Rs,      ?X,    ?X,     ?X,     0  ] -> [   0  ];
  [   C,    Host,      ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   C,    Video,     ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   C,      IO,       0,    ?X,     63,    ?X  ] -> [   0  ];
  [   C,   Special,     0,    ?X,     63,    ?X  ] -> [  ?X  ];
  [   C,   Refresh,    ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   C,     SAM,      ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   C,    Dummy,     ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   0,    Host,      ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   0,    Video,     ?X,    ?X,     63,    ?X  ] -> [   0  ];
  [   C,      IO,       1,     0,     63,    ?X  ] -> [   0  ];
  [   C,   Special,     1,    ?X,     63,    ?X  ] -> [  ?X  ];
  [   C,    Host,      ?X,    ?X,      0,    ?X  ] -> [  63  ];
  [   C,    Video,     ?X,    ?X,      0,    ?X  ] -> [  63  ];
  [   C,      IO,       0,    ?X,      0,    ?X  ] -> [  63  ];
  [   C,   Special,     0,    ?X,      0,    ?X  ] -> [  ?X  ];
  [   C,   Refresh,    ?X,    ?X,      0,    ?X  ] -> [   0  ];
  [   C,     SAM,      ?X,    ?X,      0,    ?X  ] -> [   0  ];
  [   C,    Dummy,     ?X,    ?X,      0,    ?X  ] -> [   0  ];
  [   0,    Host,      ?X,    ?X,      0,    ?X  ] -> [  63  ];
  [   0,    Video,     ?X,    ?X,      0,    ?X  ] -> [  63  ];
}
TEST_VECTORS 'Check operation of SIZE and OFFSET bits in VIRREG:'
( [ DA_SB, TYPE, MATCH, D7, D0 ] -> [ SIZE, OFFSET ] )
@IRP X (0,1)  "Perform checks twice substituting 0's then 1's for X's.
{
" Ensure they take up the correct values from reset:
  [   C,   Special,   1,1-Full,1 ] -> [ 1-Full,   1   ];
  [   C,      Rs,     ?X,  ?X, ?X ] -> [ Full,     0   ];
" Write all combinations:
  [   C,   Special,   1,    0,  0 ] -> [   0,      0   ];
  [   C,   Special,   1,    0,  1 ] -> [   0,      1   ];
  [   C,   Special,   1,    1,  0 ] -> [   1,      0   ];
  [   C,   Special,   1,    1,  1 ] -> [   1,      1   ];
```

```
" Ensure no other transactions disturb their values:
  [   C,    Host,    ?X,   ?X, ?X ] -> [  1,     1   ];
  [   C,   Video,    ?X,   ?X, ?X ] -> [  1,     1   ];
  [   C,      IO,    ?X,   ?X, ?X ] -> [  1,     1   ];
  [   C, Special,     0,   ?X, ?X ] -> [  1,     1   ];
  [   C, Refresh,    ?X,   ?X, ?X ] -> [  1,     1   ];
  [   C,     SAM,    ?X,   ?X, ?X ] -> [  1,     1   ];
  [   C,   Dummy,    ?X,   ?X, ?X ] -> [  1,     1   ];
  [   C, Special,     1,    0,  0 ] -> [  0,     0   ];
  [   C,    Host,    ?X,   ?X, ?X ] -> [  0,     0   ];
  [   C,   Video,    ?X,   ?X, ?X ] -> [  0,     0   ];
  [   C,      IO,    ?X,   ?X, ?X ] -> [  0,     0   ];
  [   C, Special,     0,   ?X, ?X ] -> [  0,     0   ];
  [   C, Refresh,    ?X,   ?X, ?X ] -> [  0,     0   ];
  [   C,     SAM,    ?X,   ?X, ?X ] -> [  0,     0   ];
  [   C,   Dummy,    ?X,   ?X, ?X ] -> [  0,     0   ];
}
TEST_VECTORS 'Check operation of DATAMS bit in VIDREG:'
" DATAMS not available directly, so have to monitor it via the EBLS and EBMS
" outputs.  To do this, assume MATCH & ( TYPE == Host ) & ( SIZE == Full ),
" then EBMS = DATAMS and EBLS = !DATAMS.
( [ DA_SB, TYPE, MATCH, D7 ] -> [ EBLS, EBMS ] )
@IRP X (0,1)  "Perform checks twice substituting 0's then 1's for X's.
{
" Check reset function:
  [   C,     IO,      1,   1 ] -> [   X,    X ];
  [   C,     Rs,     ?X,  ?X ] -> [   X,    X ];
  [   0,   Host,      1,  ?X ] -> [   1,    0 ];
" Check 0 and 1 writes:
  [   C,     IO,      1,   1 ] -> [   X,    X ];
  [   0,   Host,      1,  ?X ] -> [   0,    1 ];
  [   C,     IO,      1,   0 ] -> [   X,    X ];
  [   0,   Host,      1,  ?X ] -> [   1,    0 ];
```

```
" Ensure no other transactions disturb its value:
  [   C,    Host,   ?X,  ?X ] -> [   X,     X ];
  [   C,    Video,  ?X,  ?X ] -> [   X,     X ];
  [   C,     IO,     0,  ?X ] -> [   X,     X ];
  [   C,  Special, ?X, Full] -> [   X,     X ];
  [   C,  Refresh, ?X,  ?X ] -> [   X,     X ];
  [   C,     SAM,   ?X,  ?X ] -> [   X,     X ];
  [   C,    Dummy,  ?X,  ?X ] -> [   X,     X ];
  [   0,    Host,    1,  ?X ] -> [   1,     0 ];
  [   C,     IO,     1,   1 ] -> [   X,     X ];
  [   C,    Host,   ?X,  ?X ] -> [   X,     X ];
  [   C,    Video,  ?X,  ?X ] -> [   X,     X ];
  [   C,     IO,     0,  ?X ] -> [   X,     X ];
  [   C,  Special, ?X, Full] -> [   X,     X ];
  [   C,  Refresh, ?X,  ?X ] -> [   X,     X ];
  [   C,     SAM,   ?X,  ?X ] -> [   X,     X ];
  [   C,    Dummy,  ?X,  ?X ] -> [   X,     X  ];
  [   0,    Host,    1,  ?X ] -> [   0,     1 ];
}
TEST_VECTORS 'Check operation of data path control strobes'
( [ DA_SB, TYPE, MATCH, D7, D0, HI_AD ] -> [ EBLS, EBMS, EBTOP, EBBOT ] )
@IRP X (0,1)  "Perform checks twice substituting 0's then 1's for X's.
{
" Start with DATAMS,SIZE,OFFSET = 0,Full,0:
  [   C,      Rs,    ?X, ?X, ?X,    ?X  ] -> [   0,     0,    0,    0 ];
  [   0,      IO,     1, ?X, ?X,    ?X  ] -> [   1,     0,    0,    0 ];
  [   0,  Special,    1, ?X, ?X,    ?X  ] -> [   1,     0,    0,    0 ];
  [   0,      IO,     0, ?X, ?X,    ?X  ] -> [   0,     0,    0,    0 ];
  [   0,  Special,    0, ?X, ?X,    ?X  ] -> [   0,     0,    0,    0 ];
  [   0,    Video,    1, ?X, ?X,     0  ] -> [   1,     0,    0,    1 ];
  [   0,    Host,     1, ?X, ?X,     0  ] -> [   1,     0,    0,    1 ];
  [   0,    Video,    0, ?X, ?X,     0  ] -> [   0,     0,    0,    0 ];
  [   0,    Host,     0, ?X, ?X,     0  ] -> [   0,     0,    0,    0 ];
  [   0,    Video,    1, ?X, ?X,     1  ] -> [   1,     0,    1,    0 ];
  [   0,    Host,     1, ?X, ?X,     1  ] -> [   1,     0,    1,    0 ];
  [   0,    Video,    0, ?X, ?X,     1  ] -> [   0,     0,    0,    0 ];
  [   0,    Host,     0, ?X, ?X,     1  ] -> [   0,     0,    0,    0 ];
```

```
[   0,  Refresh, ?X,  ?X, ?X,    ?X ] -> [  0,     0,  1,    1 ];
[   0,     SAM,  ?X,  ?X, ?X,    ?X ] -> [  0,     0,  1,    1 ];
[   0,   Dummy,  ?X,  ?X, ?X,    ?X ] -> [  X,     0,  0,    0 ];
[   0,      Rs,  ?X,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
" Flip DATAMS:
[   C,      IO,   1,   1, ?X,    ?X ] -> [  0,     1,  0,    0 ];
[   0,  Special,  1,  ?X, ?X,    ?X ] -> [  1,     0,  0,    0 ];
[   0,      IO,   0,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
[   0,  Special,  0,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
[   0,    Video,  1,  ?X, ?X,     0 ] -> [  0,     1,  0,    1 ];
[   0,     Host,  1,  ?X, ?X,     0 ] -> [  0,     1,  0,    1 ];
[   0,    Video,  0,  ?X, ?X,     0 ] -> [  0,     0,  0,    0 ];
[   0,     Host,  0,  ?X, ?X,     0 ] -> [  0,     0,  0,    0 ];
[   0,    Video,  1,  ?X, ?X,     1 ] -> [  0,     1,  1,    0 ];
[   0,     Host,  1,  ?X, ?X,     1 ] -> [  0,     1,  1,    0 ];
[   0,    Video,  0,  ?X, ?X,     1 ] -> [  0,     0,  0,    0 ];
[   0,     Host,  0,  ?X, ?X,     1 ] -> [  0,     0,  0,    0 ];
[   0,  Refresh, ?X,  ?X, ?X,    ?X ] -> [  0,     0,  1,    1 ];
[   0,     SAM,  ?X,  ?X, ?X,    ?X ] -> [  0,     0,  1,    1 ];
[   0,   Dummy,  ?X,  ?X, ?X,    ?X ] -> [  0,     X,  0,    0 ];
[   0,      Rs,  ?X,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
" Now DATAMS,SIZE,OFFSET = 0,Full,1:
[   C,      Rs,  ?X,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
[   C,  Special,  1, Full, 1,    ?X ] -> [  1,     0,  0,    0 ];
[   0,      IO,   1,  ?X, ?X,    ?X ] -> [  1,     0,  0,    0 ];
[   0,  Special,  1,  ?X, ?X,    ?X ] -> [  1,     0,  0,    0 ];
[   0,      IO,   0,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
[   0,  Special,  0,  ?X, ?X,    ?X ] -> [  0,     0,  0,    0 ];
[   0,    Video,  1,  ?X, ?X,     0 ] -> [  1,     0,  0,    1 ];
[   0,     Host,  1,  ?X, ?X,     0 ] -> [  1,     0,  0,    1 ];
[   0,    Video,  0,  ?X, ?X,     0 ] -> [  0,     0,  0,    0 ];
[   0,     Host,  0,  ?X, ?X,     0 ] -> [  0,     0,  0,    0 ];
[   0,    Video,  1,  ?X, ?X,     1 ] -> [  1,     0,  1,    0 ];
[   0,     Host,  1,  ?X, ?X,     1 ] -> [  1,     0,  1,    0 ];
[   0,    Video,  0,  ?X, ?X,     1 ] -> [  0,     0,  0,    0 ];
[   0,     Host,  0,  ?X, ?X,     1 ] -> [  0,     0,  0,    0 ];
[   0,  Refresh, ?X,  ?X, ?X,    ?X ] -> [  0,     0,  1,    1 ];
```

```
  [ 0,    SAM,    ?X,  ?X, ?X,    ?X ] -> [  0,    0,  1,   1 ];
  [ 0,    Dummy,  ?X,  ?X, ?X,    ?X ] -> [  X,    0,  0,   0 ];
  [ 0,     Rs,    ?X,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
" Flip DATAMS:
  [ C,     IO,     1,   1, ?X,    ?X ] -> [  0,    1,  0,   0 ];
  [ 0,  Special,   1,  ?X, ?X,    ?X ] -> [  1,    0,  0,   0 ];
  [ 0,     IO,     0,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
  [ 0,  Special,   0,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
  [ 0,   Video,    1,  ?X, ?X,     0 ] -> [  0,    1,  0,   1 ];
  [ 0,   Host,     1,  ?X, ?X,     0 ] -> [  0,    1,  0,   1 ];
  [ 0,   Video,    0,  ?X, ?X,     0 ] -> [  0,    0,  0,   0 ];
  [ 0,   Host,     0,  ?X, ?X,     0 ] -> [  0,    0,  0,   0 ];
  [ 0,   Video,    1,  ?X, ?X,     1 ] -> [  0,    1,  1,   0 ];
  [ 0,   Host,     1,  ?X, ?X,     1 ] -> [  0,    1,  1,   0 ];
  [ 0,   Video,    0,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,   Host,     0,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,  Refresh,  ?X,  ?X, ?X,    ?X ] -> [  0,    0,  1,   1 ];
  [ 0,    SAM,    ?X,  ?X, ?X,    ?X ] -> [  0,    0,  1,   1 ];
  [ 0,   Dummy,   ?X,  ?X, ?X,    ?X ] -> [  0,    X,  0,   0 ];
  [ 0,     Rs,    ?X,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
" Now DATAMS,SIZE,OFFSET = 0,Half,0:
  [ C,     Rs,    ?X,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
  [ C,  Special,   1, Half, 0,    ?X ] -> [  1,    0,  0,   0 ];
  [ 0,     IO,     1,  ?X, ?X,     0 ] -> [  1,    0,  0,   0 ];
  [ 0,     IO,     1,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,  Special,   1,  ?X, ?X,    ?X ] -> [  1,    0,  0,   0 ];
  [ 0,     IO,     0,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
  [ 0,  Special,   0,  ?X, ?X,    ?X ] -> [  0,    0,  0,   0 ];
  [ 0,   Video,    1,  ?X, ?X,     0 ] -> [  1,    0,  1,   0 ];
  [ 0,   Host,     1,  ?X, ?X,     0 ] -> [  1,    0,  1,   0 ];
  [ 0,   Video,    0,  ?X, ?X,     0 ] -> [  0,    0,  0,   0 ];
  [ 0,   Host,     0,  ?X, ?X,     0 ] -> [  0,    0,  0,   0 ];
  [ 0,   Video,    1,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,   Host,     1,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,   Video,    0,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,   Host,     0,  ?X, ?X,     1 ] -> [  0,    0,  0,   0 ];
  [ 0,  Refresh,  ?X,  ?X, ?X,    ?X ] -> [  0,    0,  1,   1 ];
```

```
[   0,     SAM,    ?X,   ?X, ?X,    ?X ] -> [   0,    0,   1,    1 ];
[   0,   Dummy,   ?X,   ?X, ?X,    ?X ] -> [   X,    0,   0,    0 ];
[   0,     Rs,     ?X,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
" Flip DATAMS:
[   C,     IO,      1,    1, ?X,     0 ] -> [   0,    1,   0,    0 ];
[   0,     IO,      1,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
[   0,  Special,    1,   ?X, ?X,    ?X ] -> [   1,    0,   0,    0 ];
[   0,     IO,      0,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
[   0,  Special,    0,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
[   0,   Video,     1,   ?X, ?X,     0 ] -> [   0,    1,   1,    0 ];
[   0,    Host,     1,   ?X, ?X,     0 ] -> [   0,    1,   1,    0 ];
[   0,   Video,     0,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,    Host,     0,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,   Video,     1,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
[   0,    Host,     1,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
[   0,   Video,     0,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
[   0,    Host,     0,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
[   0,  Refresh,   ?X,   ?X, ?X,    ?X ] -> [   0,    0,   1,    1 ];
[   0,     SAM,    ?X,   ?X, ?X,    ?X ] -> [   0,    0,   1,    1 ];
[   0,   Dummy,   ?X,   ?X, ?X,    ?X ] -> [   0,    X,   0,    0 ];
[   0,     Rs,     ?X,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
" Finally DATAMS,SIZE,OFFSET = 0,Half,1:
[   C,     Rs,     ?X,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
[   C,  Special,    1, Half,  1,    ?X ] -> [   1,    0,   0,    0 ];
[   0,     IO,      1,   ?X, ?X,     1 ] -> [   1,    0,   0,    0 ];
[   0,     IO,      1,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,  Special,    1,   ?X, ?X,    ?X ] -> [   1,    0,   0,    0 ];
[   0,     IO,      0,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
[   0,  Special,    0,   ?X, ?X,    ?X ] -> [   0,    0,   0,    0 ];
[   0,   Video,     1,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,    Host,     1,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,   Video,     0,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,    Host,     0,   ?X, ?X,     0 ] -> [   0,    0,   0,    0 ];
[   0,   Video,     1,   ?X, ?X,     1 ] -> [   1,    0,   1,    0 ];
[   0,    Host,     1,   ?X, ?X,     1 ] -> [   1,    0,   1,    0 ];
[   0,   Video,     0,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
[   0,    Host,     0,   ?X, ?X,     1 ] -> [   0,    0,   0,    0 ];
```

```
  [   0,  Refresh, ?X,    ?X, ?X,     ?X  ] -> [   0,     0,  1,    1  ];
  [   0,     SAM,  ?X,    ?X, ?X,     ?X  ] -> [   0,     0,  1,    1  ];
  [   0,   Dummy,  ?X,    ?X, ?X,     ?X  ] -> [   X,     0,  0,    0  ];
  [   0,      Rs,  ?X,    ?X, ?X,     ?X  ] -> [   0,     0,  0,    0  ];
" Flip DATAMS:
  [   C,      IO,   1,     1, ?X,      1  ] -> [   0,     1,  0,    0  ];
  [   0,      IO,   1,    ?X, ?X,      0  ] -> [   0,     0,  0,    0  ];
  [   0, Special,   1,    ?X, ?X,     ?X  ] -> [   1,     0,  0,    0  ];
  [   0,      IO,   0,    ?X, ?X,     ?X  ] -> [   0,     0,  0,    0  ];
  [   0, Special,   0,    ?X, ?X,     ?X  ] -> [   0,     0,  0,    0  ];
  [   0,   Video,   1,    ?X, ?X,      0  ] -> [   0,     0,  0,    0  ];
  [   0,    Host,   1,    ?X, ?X,      0  ] -> [   0,     0,  0,    0  ];
  [   0,   Video,   0,    ?X, ?X,      0  ] -> [   0,     0,  0,    0  ];
  [   0,    Host,   0,    ?X, ?X,      0  ] -> [   0,     0,  0,    0  ];
  [   0,   Video,   1,    ?X, ?X,      1  ] -> [   0,     1,  1,    0  ];
  [   0,    Host,   1,    ?X, ?X,      1  ] -> [   0,     1,  1,    0  ];
  [   0,   Video,   0,    ?X, ?X,      1  ] -> [   0,     0,  0,    0  ];
  [   0,    Host,   0,    ?X, ?X,      1  ] -> [   0,     0,  0,    0  ];
  [   0, Refresh,  ?X,    ?X, ?X,     ?X  ] -> [   0,     0,  1,    1  ];
  [   0,     SAM,  ?X,    ?X, ?X,     ?X  ] -> [   0,     0,  1,    1  ];
  [   0,   Dummy,  ?X,    ?X, ?X,     ?X  ] -> [   0,     X,  0,    0  ];
  [   0,      Rs,  ?X,    ?X, ?X,     ?X  ] -> [   0,     0,  0,    0  ];
}
TEST_VECTORS 'Register write and external reset REGSB & RESET'
( [ DA_SB, TYPE, MATCH, LO_AD ] -> [ REGSB, RESET ] )
@IRP X (0,1)  "Perform checks twice substituting 0's then 1's for X's.
{
  [    0,      IO,      1,    1   ] -> [  0,       0  ];
  [    1,      IO,      1,    1   ] -> [  1,       0  ];
  [   ?X,      IO,      0,    1   ] -> [  0,       0  ];
  [   ?X,      IO,      1,    0   ] -> [  0,       0  ];
  [   ?X,    Host,      1,    1   ] -> [  0,       0  ];
  [   ?X,   Video,      1,    1   ] -> [  0,       0  ];
  [   ?X, Special,      1,    1   ] -> [  0,       0  ];
  [   ?X, Refresh,      1,    1   ] -> [  0,       0  ];
  [   ?X,     SAM,      1,    1   ] -> [  0,       0  ];
  [   ?X,   Dummy,      1,    1   ] -> [  0,       0  ];
```

```
[    1,    Rs,    ?X,   ?X   ] -> [   0,     1   ];
[    0,    Rs,    ?X,   ?X   ] -> [   0,     0   ];
}
END fsumach1;
```

FITTER FILE

Note: This file has been condensed in order to save trees.

```
(C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990,1991
Flags Used:             Unplace=False        Max Packing=True
Flags Used:         Expand Small=True          Expand All=False
*********************************************************************************
     Mach PLD Fitter - v 1.51          Bus Access Controller for the DIS4000
                                        Framestore Unit
*********************************************************************************
*** Timing Analysis for Signals
 Parameter    Min    Max            Signal List (Those having Max delay.)
      Tpd      1      1                  C0              C1             C2
                                         C3              C4             C5
                                      EBTOP           RESET
      Tsu      1      1              OFFSET            VID0           VIR0
                                       VID1            VIR1           VID2
                                       VIR2          DATAMS
      Tco      0      1               EBTOP           EBBOT           EBLS
                                       EBMS              C0             C1
                                         C2              C5
      Tcr      1      1                SIZE          OFFSET
.
 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - clock to output (register output to combinatorial output)
 Tcr - Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                 Available         Used         Remaining
        Clocks:      2              1              1
          Pins:     38             35              3      ->     92%
     I/O Macro:     32             14             18
   Total Macro:     32             27              5
 Product Terms:    128             84             20      ->     84%
MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
```

```
*** Last Equations Placed in Blocks

Weakly -

Assign -

*** Block Partitioning Results

              Array    Macros     # I/O    Buried   Product    Signal
              Inputs   Remain     Macro    Logic    Terms      Fanout

   Block-> A   20        1         10        5       60          7
   Block-> B   21        4          4        8       48          8

*** Block Signal List

Block-> A         OFFSET         SIZE         DATAMS          VIR5
                  VIR0           VID5         VID0            RESET
                  REGSB          EBMS         EBLS            EBBOT
                  EBTOP          C5           C0

Block-> B         VIR4           VIR3         VIR2            VIR1
                  VID4           VID3         VID2            VID1
                  C4             C3           C2              C1

|> INFORMATION F050 - Device Utilization....... *: 88 %
```

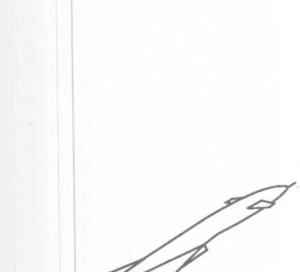
```
*** Feedback Map - Bus Access Controller for the DIS4000 Framestore Unit
 Gbl Inp .--.      I/O   .--+--A--+--.  I/O          I/O  .--+--B--+--.  I/O
         | 0|            | 0|     |21| MATCH          P4 : 0|     |21| MATCH
         | 1|   OFFSET : 1|     |20| T2           VIR1 : 1|     |20| T2
         | 2|            | 2|   |19| T1             P3 : 2|     |19| T1
         | 3|     SIZE : 3|     |18| DA_SB        VIR2 : 3|     |18|
         | 4|    HI_AD : 4|     |17- T0             P2 : 4|     |17- T0
         | 5|   DATAMS : 5|     |16- LO_AD        VIR3 : 5|     |16- LO_AD
         '--'       D0 : 6|     |15: P0             P1 : 6|     |15: VID4
                  VIR0 : 7|     |14: P5           VIR4 : 7|     |14: D4
                    D1 : 8|     |13: VID5           D2 : 8|     |13: VID3
                  VIR5 : 9|     |12: D7           VID1 : 9|     |12: D3
                    D6 :10|     |11: VID0           D5 :10|     |11: VID2
K: - Dedicated Input   '--+-u--u+--'                  '--+-u--u+--'
*** Logic Map - Bus Access Controller for the DIS4000 Framestore Unit
 Gbl Inp .--.      I/O   .--+--A--+--.  I/O          I/O  .--+--B--+--.  I/O
   LO_AD| 0|    /RESET | 0| 1    |21|                C1 | 0| 3    |21|
      T0| 1|    OFFSET | 1| 3    |20|              VIR1 | 1| 4    |20|
      T1| 2|     /EBLS | 2| 4    |19|                C2 | 2| 3    |19|
      T2| 3|      SIZE | 3| 3    |18|              VIR2 | 3| 4    |18|
   MATCH| 4|     /EBMS | 4| 3    |17|                C3 | 4| 3    |17|
   DA_SB| 5|    DATAMS | 5| 3    |16|              VIR3 | 5| 4    |16|
        '--'    /EBBOT | 6| 2   .|15|                C4 | 6| 3   3|15| VID4
                  VIR0 | 7| 4   4|14| /EBTOP       VIR4 | 7| 4   .|14|
                    C0 | 8| 3   3|13| VID5              | 8| .   3|13| VID3
                  VIR5 | 9| 4   3|12| C5           VID1 | 9| 3   .|12|
                /REGSB |10| 1   3|11| VID0              |10| .   3|11| VID2
K: - Dedicated Input   '--+-u--u+--'                  '--+-u--u+--'
```

```
*** Pin Map - Bus Access Controller for the DIS4000 Framestore Unit
                          /RESET
                      OFFSET  |         D1
                    /EBLS  |  |         |  .
                  SIZE  |  |  |         |  |  D5
              /EBMS  |  |  |  |         |  |  |  .
                  |  |  |  |  |         |  |  |  |
            .-----'--'--'--'--'--o-----'--'--'--'---.
            |                        4  4  4  4  4  |
            |     6  5  4  3  2  1  4  3  2  1  0   |
         D0| 7                                    39|D3
     /EBBOT| 8                      G  V          38|
      HI_AD| 9                      n  c          37|P5
      LO_AD|10                      d  c          36|P0
         T0|11                                    35|DA_SB
       Gnd |12             MACH-110               34| Gnd
         T1|13                                    33|MATCH
         C0|14                V  G                32|T2
         D6|15                c  n                31|P4
     /REGSB|16                c  d                30|C4
         D2|17                                    29|P3
            |   1  1  2  2  2  2  2  2  2  2  2   |
            |   8  9  0  1  2  3  4  5  6  7  8   |
            '---.--.--.--.-------.--.--.--.--.----'
                |  |  |  |       |  |  |  |
               C5  |  |  |       |  |  |  |  C3
                  D4  |  |       |  |  |  P2
                 /EBTOP  |       |  |  C2
                        D7       |  P1
                                 C1
The Design Doc is  stored in ===> fsumach1.Rpt
The Jedec Data is  stored in ===> fsumach1.Jed
The Placements are stored in ===> fsumach1.Plc
The  Fuse Plot is  stored in ===> fsumach1.Xpt
%% FITR %% Error Count: 0, Warning Count: 2
%% FITR %% File Processed Successfully. - File: fsumach1
```

PLACEMENT FILE

```
$DEVICE MACH110 fit
$PINS 0
lace=False          Max Packing=True
; Flags Used:          Expand Small=True          Expand All=False
; FITR generated placements
Pin    7              D0  ; Inp         ; A  5
Pin   43              D1  ; Inp         ; B 15
Pin   17              D2  ; Inp         ; A 11
Pin   39              D3  ; Inp         ; B 11
Pin   19              D4  ; Inp         ; A 13
Pin   41              D5  ; Inp         ; B 13
Pin   15              D6  ; Inp         ; A  9
Pin   21              D7  ; Inp         ; A 15
Pin   36              P0  ; Inp         ; B  8
Pin   25              P1  ; Inp         ; B  1
Pin   27              P2  ; Inp         ; B  3
Pin   29              P3  ; Inp         ; B  5
Pin   31              P4  ; Inp         ; B  7
Pin   37              P5  ; Inp         ; B  9
Pin   33           MATCH  ; Inp         ; I  4
Pin   35           DA_SB  ; Inp         ; I  5
Pin    9           HI_AD  ; Inp         ; A  7
Pin   10           LO_AD  ; Inp         ; I  0
Pin   11              T0  ; Inp         ; I  1
Pin   13              T1  ; Inp         ; I  2
Pin   32              T2  ; Inp         ; I  3
Pin   14              C0    Comb        ; A  8
Pin   24              C1    Comb        ; B  0
Pin   26              C2    Comb        ; B  2
Pin   28              C3    Comb        ; B  4
Pin   30              C4    Comb        ; B  6
Pin   18              C5    Comb        ; A 12
Node   5          SIZE_I     Reg        ; A  3
Node   3        OFFSET_I     Reg        ; A  1
Pin   20          /EBTOP    Comb        ; A 14
Pin    8          /EBBOT    Comb        ; A  6
```

```
Pin     4           /EBLS   Comb          ; A  2
Pin     6           /EBMS   Comb          ; A  4
Pin    16          /REGSB   Comb          ; A 10
Pin     2          /RESET   Comb          ; A  0
Node   13            VID0    Reg          ; A 11
Node   27            VID1    Reg          ; B  9
Node   29            VID2    Reg          ; B 11
Node   31            VID3    Reg          ; B 13
Node   33            VID4    Reg          ; B 15
Node   15            VID5    Reg          ; A 13
Node    9            VIR0    Reg          ; A  7
Node   19            VIR1    Reg          ; B  1
Node   21            VIR2    Reg          ; B  3
Node   23            VIR3    Reg          ; B  5
Node   25            VIR4    Reg          ; B  7
Node   11            VIR5    Reg          ; A  9
Node    7          DATAMS    Reg          ; A  5
Pin     5            SIZE    Reg          ; A  3
Pin     3          OFFSET    Reg          ; A  1
; Group Mach_Seg_A OFFSET SIZE DATAMS VIR5 VIR0 VID5 VID0 RESET REGSB EBMS EBLS EB-
BOT EBTOP C5 C0
; Group Mach_Seg_B VIR4 VIR3 VIR2 VIR1 VID4 VID3 VID2 VID1 C4 C3 C2 C1
```



Psion Organizer Digital Interface: MACH110

Advanced Micro Devices

Application Note

by Kamru Miah, MAFF Central Science Library, United Kingdom

INTRODUCTION

Capturing of data from field experiment requires portable data acquisition systems which should be low powered, high speed, light weight, compact size and low cost. Such a system could be developed using a handheld microcomputer linked to a Digital Interface (DI), incorporating low powered ASIC technology components to meet the above criteria. This article describes the design and development of a 16 input and a 16 output DI circuit, using the MACH110 device as the controller of the interface. It is linked to the Psion Organizer which controls the overall process.

RESOURCES AVAILABLE

The top slot of the Organizer has a 16 pin I/O port which is normally used for the serial link (to communicate with external devices). Technical information on the Organizer showed that this port could be configured to control external hardware. The pin functions of the top slot are shown below.

8	data lines (bi-directional)
3	power lines (including GROUND)
4	control lines
1	port select line

All the above pins are assigned to perform specific functions by the Organizer and are not available for use for any other purpose, except a single control (clock) line which is programmable.

DESIGN REQUIREMENTS

The Organizer is required to communicate with 32 I/O lines of the DI, but from the above pin functions it can be seen that the Organizer has only 8 I/O lines at its disposal. Therefore, the only way it can fulfill this requirement is by partitioning the 32 I/O lines into four channels of 8 data lines. It could then communicate with each channel in a sequence. The channel allocation of the DI is shown below.

Channel 1 = 8-bit Input (D0 - D7)
Channel 2 = 8-bit Input (D8 - D15)
Channel 3 = 8-bit Output (D16 - D23)
Channel 4 = 8-bit Output (D24 - D31)

In order for the Organizer to select any one of the above 4 channels, it needs to have at least 2 address lines. Since there are no address lines available, the external hardware has to have the intelligence to allow the Organizer to read the 16 inputs and write to the 16 outputs, in a sequence, using the single control line mentioned above.

To read or write from or to a particular channel, the Organizer has to select that channel by passing a unique channel selection address to the DI before a read/write operation is carried out. The address information has to be held on-board until overwritten by the next address for another read/write operation. Also when the Organizer moves from write mode to read mode, the data written on the particular output channel has to be latched until overwritten by another write operation. This is important for using the DI in a closed-loop control application. These considerations mean that the DI must be required to latch both the address and the output data while the Organizer carries out any other task/operation.

It is also desirable for the DI to hold its control software on-board in an EPROM, which can be loaded into the Organizer's memory when it "boots up." This technique would allow the user to have more space to store the data on the spare Data Pack slots, as well as not having the need to carry a separate Data Pack with the DI. This is added in parallel with the rest of the DI circuitry, but the control logic has to disable the EPROM when communicating with the DI and vice versa. The programmable control (clock) line of the Organizer has to be used to pass the address of the EPROM as well.

DESIGN IMPLEMENTATION

In order to implement the complete design to fulfill the above requirements, the individual functional blocks were first identified. It was then decided which of functional blocks were best implemented within the MACH110 device (from a resource availability point of view) and which were best left outside to be implemented using conventional logic.

Since both the data and the address from the Organizer were passed to the DI via the same Data Bus, it was important to separate the address from the data by the DI circuitry. This was achieved by using the single programmable control line, MsClk, to inform the DI whether the information on the Organizer's I/O port was data or address. If it was data, then the bi-directional Data Buyer would be enabled, and if it was an address, then the address buffer would be enabled. The channel selection address would then be latched. From the channel selection address, the direction of the Data Bus would be determined. The Organizer would then be expected to raise MsClk "high," write the channel selection

Publication# 17749 Rev. A Amendment/0
Issue Date: May 1993

address and then "lower" MsClk before performing the read/write operation. That is, if the Organizer had selected channel 1 or 2, then it would be expected to read data from the port, and if it had selected channel 3 or 4, then it would be expected to write data to the port.

The read/write operation of the Psion Organizer was investigated by writing a test program in assembly language and developing the hardware in conventional logic, using TTL ICs. It was found that from the instant MsClk changed from "low" to "high," it took between 52 to 95 uS before the write data appeared at the output port. The minimum pulse width of the data was found to be about 15 uS, before the data disappeared from the port again. If MsClk was lowered immediately after it was raised, its pulse width was found to be about 7 uS. In light of these facts, it was deduced that from the instant MsClk changed from "low" to "high" it was necessary to latch the channel address decoder after 5 uS and the output latches of channel 3 and 4 after 105 uS, respectively.

A delay circuit was therefore required which would generate these "one-shot" pulses upon receiving the raising edge of MsClk. This delay circuit should then remain idle until another read/write operation initiated a further raising edge of MsClk. The delay circuit could be implemented using a 7-stage binary counter, controlled by a single D-type flip-flop, to start and stop the counter.

The next step was to carry out a detailed design of the complete system and determine how many of the functions could actually be included inside the MACH110 device (in terms of number of I/O pins, clock inputs etc.). The input multiplexers, the output latches, and the EPROM circuit could be implemented using conventional logic, and the rest of the functions could be included inside the MACH110 device. The functions shown below were identified initially for implementation within the MACH110 device.

BusTxRx—Octal bus transceiver with tri-state outputs, including 3-bit Address Buffer.

BusEnb—Combinational logic for enabling the Data Buffer OR the Address Buffer (within BusTxRx).

BinCnt—Synchronous counter for delayed pulse generation, including start/stop control—used for latching purposes.

Ch_decod—Channel decoder—latches the address from the Organizer and selects the appropriate channel.

Each of the above functions was implemented as a separate module (in ABEL4 language) and simulated for design validation. They were then combined into a main source file and simulated for design validation, as before. Appropriate inputs, outputs, sets, constants, equations, and test vectors were defined in each case as necessary. When the fitting of the main program into the MACH110 device was tried (using SmartPart fitter) it was not successful. This was due to the occurrence of fitter errors, such as (a) Too many product terms found, (b) No TRST available on buried nodes, (c) Too many clocks in user design, (d) Connection problem (wiring congested) etc.

The source of the errors was identified and the initial design was modified as necessary to obtain a successful fit. The modifications consisted of amending the design equations and removing the binary counter and the bus transceiver, BinCnt and BusTxRx, respectively, outside the MACH110 device. The 7-stage binary counter function was easily implemented using a single CMOS IC (4024). This modification removed the need for an extra clock input to the MACH110 device, which was not available. Similarly, the octal bus transceiver function was easily implemented externally using a single TTL IC (74LS245). This reduced the number of product terms and I/O pins used within the design making it much easier for the SmartPart fitter to program the Switch Matrix, without raising any wiring congested errors.

HARDWARE DESIGN

Before the Organizer can communicate via its top slot (slot 3) to the DI, it must first set SelS3 signal "low" and SOEnb signal "high." (SOEnb is set "low" when it is in Data Pack mode.) The DI must not be enabled unless this condition is met. The programmable control (clock) line, MsClk, is used by the Organizer to pass the EPROM address when it is in Data Pack mode. It is not used by the Organizer when it is in Hardware mode. Therefore, MsClk was used to inform the DI (Hardware) that the information on the Organizer Bus was an address when it was set "high" and Data when it was set "low."

The complete circuit diagram of the 16 input and 16 output DI to the Psion Organizer, using the MACH110 device as the controller, is shown in Figure 2-1. IC1 is the bi-directional (octal) bus transceiver which is enabled when a read/write operation is performed from or to the input/output channels. Its direction is determined from the channel selection address passed previously to the DI by the Organizer when in Addressing Mode. IC2 is an octal buffer with tristate outputs. The top half is used as the Address Buffer—which is enabled when the Organizer passes the channel selection address to the DI. The bottom half is permanently enabled and is used to simply buffer the control signals before connecting them to the rest of the circuit. Both ICI and IC2 have been used, apart from performing their buffering functions, to provide a degree of protection to the internal components, particularly the MACH110 device, from the "outside" world. This is one of the main reason for not including these functions within the MACH110 device.

IC3 is a 12-stage binary counter which acts as an address decoder to IC4. IC4 is a 16 kbyte EPROM which is

used to hold the control program of the DI on-board. The program is loaded into the Organizer's memory at boot-up. IC5 is the MACH110 device which controls the reliable operation of the complete circuit. It will be explained in detail, later. IC6 is the 7-stage binary counter which is used by IC5 to calculate various delays for latching purposes, as explained before. It is driven by a 1 MHz system clock, which could be implemented using a crystal controlled oscillator IC (such as 74S124).

IC7 and IC8 are quad 2-to-1 multiplexers, which are enabled when the Organizer wishes to carry out a read operation from channel 1 or channel 2. When the decoded signal, RSel, from IC5 is "high," channel 1 is selected and when it's "low," channel 2 is selected. IC9 and IC10 are octal D-type edge-triggered flip-flops. They are latched by WClk1 or WClk2, depending on which output channel the Organizer wishes to write the data to. That is, if the data is to be written to channel 3, then IC9 will be latched after the data has appeared on the Data Bus (INTBUSI). Similarly, if the data is to be written to channel 4, then IC10 will be latched.

MACH110 LOGIC DESIGN

Having explained the functionality of the associated components around the MACH110 device, it is now possible to explain the logic functions the MACH110 device has to perform in order to generate the required control signals (see Figure 2-1). The enable signals for IC1 and IC2 have been named Enbl and Enb2, respectively, which are described by the following Boolean equations in ABEL4-HDL.

```
!Enb1 = !SelS3 & SOEnb & !MsClk ;      2-1
!Enb2 = !SelS3 & SOEnb & !MsClk ;      2-2
```

The above equations form part of the *BusEnb* module mentioned before. Since the clock input signal of the EPROM's *address decoder* IC uses the same control signals, this equation was included in the above module as well. The clock input is named CP, and the equation is as follows:

```
CP = !((!SelS3&SOEnb) # !MsClk) ;      2-3
```

It can be seen from the above equation that when the Organizer is in Data Pack mode (ie. SelS3 = O and SOEnb = 1) then CP = MsClk. Hence, by-passing the clock signal to the EPROM's address decoder IC. Otherwise CP will remain "low" and MsClk will be used by the DI for another purpose, as explained before.

The BinCnt module was then designed to obtain the delay signals used in the Ch_Decod module. Since a synchronous counter was being used for generation of the delay signals (rather then monostables), it was necessary to stop the counter from re-generating the delay pulses.

This was achieved by controlling the Reset input of the counter. That is, the delay pulses were generated only when the Reset input was "low." Otherwise it was always kept "high." This technique is illustrated by the following Boolean equations:

```
Reset1.D = I;                  2-4
Reset1.C = MsClk;              2-5
Reset1.AR = Reset2;            2-6
Reset2 := (Dcnt == 127);       2-7
Dcnt := (Dcnt + 1) & !Reset1;  2-8
Dcnt.c = SysClk;               2-9
Dcnt.AR = Reset1;              2-10
DlyClk1 = (Dcnt == 5);         2-11
DlyClk2 = (Dcnt == 105);       2-12
```

Reset1 in the above equations was defined as a single D-type flip-flop (with inverted output) and Dcnt was defined as a 7-stage binary counter. Equation 2-4 shows that the data input of Reset1 was always kept "high" and when Dcnt reached maximum count (Equation 2-7), it resetted the output of Reset1 flip-flop. The !Q output of Reset1 in turn resetted Dcnt and held it in that state until a positive edge was reached at the clock input of Reset1 (Equation 2-5). When MsClk changes from "low" to "high" (data mode to address mode) it causes the !Q output of Reset1 to go "low," starting the counter at a rate set by the system clock, SysClk. Since the frequency of SysClk is 1 MHz, the outputs of DlyClkl1 and DlyClk2 are 5 and 105 uS, respectively. This process would repeat itself every time a positive going edge is received from MsClk, regardless of its duration.

The Ch_Decod module comprised of three D-type flip-flops to latch the channel selection address from the Organizer before it carried out a read/write operation. Only 2-bit addressing was necessary to select any one of the 4 channels, but 3-bit addressing was chosen to introduce an idling state. F in the following equation is the set of 3 flip-flops (F2,F1,F0) and A is the set of 3 address lines (A2,A1,A0).

```
F := A;          2-13
F.C = DecClk;    2-14
```

It was decided to latch the channel address after 5 uS from the instant of MsClk changed from "low" to "high." This could have been achieved using DlyClk1, but since a product term can not be used to latch any D-type flip-flops within the MACH110 device, a different clock signal, DecClk, had to be used. Since DlyClk1 had to be used to latch the fiip-flops, it was declared as an output in the main source file (see later) and then fed back into the MACH110 device as an (clock) input under the new name, DecClk. This is illustrated in Figure 2-1.

In order to generate the rest of the control signals from the latched channel address, the appropriate outputs were assigned to various combinations of F2, F1 and F0. Table 2-1 shows the channel allocation.

Table 2-1. Channel Allocation

Latched Address			
F2	**F1**	**F0**	**Channel Name**
0	0	0	Idle
0	0	1	CH1
0	1	0	CH2
0	1	1	CH3
1	0	0	CH4

From the above table the following equations were derived.

!Dir = (!F2 & F1 & F0) # (F2 & !F1 & !F0); 2-15
!REnb = (!F2 & (!F1 # !F0)); 2-16
RSel = (!F2 & !F1 & F0); 2-17
RClk = Dir & DlyClk1; 2-18
WClkl1= (!F2 & F1 & F0) & DlyClk2; 2-19
WClk2 = (F2 & !F1 & !F0) & DlyClk2; 2-20

The Dir signal in the above equation determines the direction of IC1, in Figure 2-1. When the Organizer wishes to write any data to channel 3 or channel 4, the Dir signal causes IC1 to pass data from the DATA_BUS to the INT_BUS1; and vice versa. Similarly, when it wishes to read any data from channel 1 or channel 2, the REnb signal enables IC7 and IC8. The RSel signal selects which of the two channels read data from. The WClkl1or WClk2 signal ensures that channel 3 or channel 4 output is latched only when that particular channel is selected. The RClk signal has been implemented to provide an option to latch the input data when reading from channel 1 or channel 2. It has not been used in this design.

The complete design file in ABEL4-HDL begins on page 2-6. The input and output signals mentioned previously have been defined very clearly in the source file. The specific pin numbers shown in the declaration part of the design file were for the convenience of the printed circuit board (PCB) routing. The unassigned pin numbers were left to the SmartPart fitter to assign according to the ease of the internal interconnection strategy of the MACH110 device. All the internal I/O pins have been labeled as nodes, to keep the resource utilization of the device to a minimum; especially when the associated signals were not required externally.

This design required the individual reset of the D-type flip-flops, but the Asynchronous Reset (AR) input signal within the MACH110 device was common to all the flip-flops. This constraint gave rise to a design problem, since it was not desirable to reset the address decoder when resetting the binary counter. Further information from the manufacturer (AMD) revealed that this AR signal was common to each of the PAL blocks within the device, rather than the whole device. Therefore, to force the SmartPart fitter to group the address decoder flip-flops, (F2, F1, F0) into one PAL block and the Reset 1 flip-flop into another, the AMD MACH property command was used, as shown in the design file. The grouping of the rest of the functions were left to the fitter to decide, since they were not critical. This technique not only solved the above problem but it also made the automatic configuration of the Switch Matrix much easier, since it achieved 100% fit at the first attempt.

All the equations in the design file have already been explained, except that the equations within Bus_Enb part has been modified, using DeMorgan's theorem. Since the binary counter has been implemented outside the MACH110 device, the BinCnt part has been renamed as Delay, having removed Equations 2-8, 2-9 and 2-10, as mentioned before, from the main design file.

All the above equations have been simulated under ABEL4 environment with appropriate test sectors and were found to be satisfactory.

SUMMARY

During the course of this design, the use of the ABEL4 design software had to be learned and was found very easy to use. The user manual provided with the software was very helpful and had sufficient examples to aid the understanding of the ABEL Hardware Description Language fairly rapidly. The compile and simulation feature of the software was extremely useful, since it allowed rapid design validation of functional blocks, independent of any device type or pin assignments. The .fit file produced by the SmartPart fitter was very useful in producing a design to fit. It provided useful information regarding the resource availability and other features of the device which may have been over-looked by the designer.

Although the initial design incorporating the bus transceiver was unable to be fitted within the MACH110 device, it was found to fit within the MACH210 device since it has extra buried nodes. However, implementing this function using a standard TTL IC was found to be quite satisfactory since it would provide a degree of protection to the MACH110. Similarly, the functionality of IC7 –IC10 in Figure 2-1 were best implemented using standard TTL/CMOS ICs rather than using a higher density MACH device, for the same reason.

Although the MACH110 device used in this design was for the purpose of "MACH CHALLENGE" competition, it is quite appropriate for inclusion within a final product, since this device has sufficient density to implement all the random logic functions of this design, reducing the PCB size and component counts by approximately 40 to 50% compared to conventional design; which meets the low power, high speed, compact size, and light weight requirements quite satisfactorily.

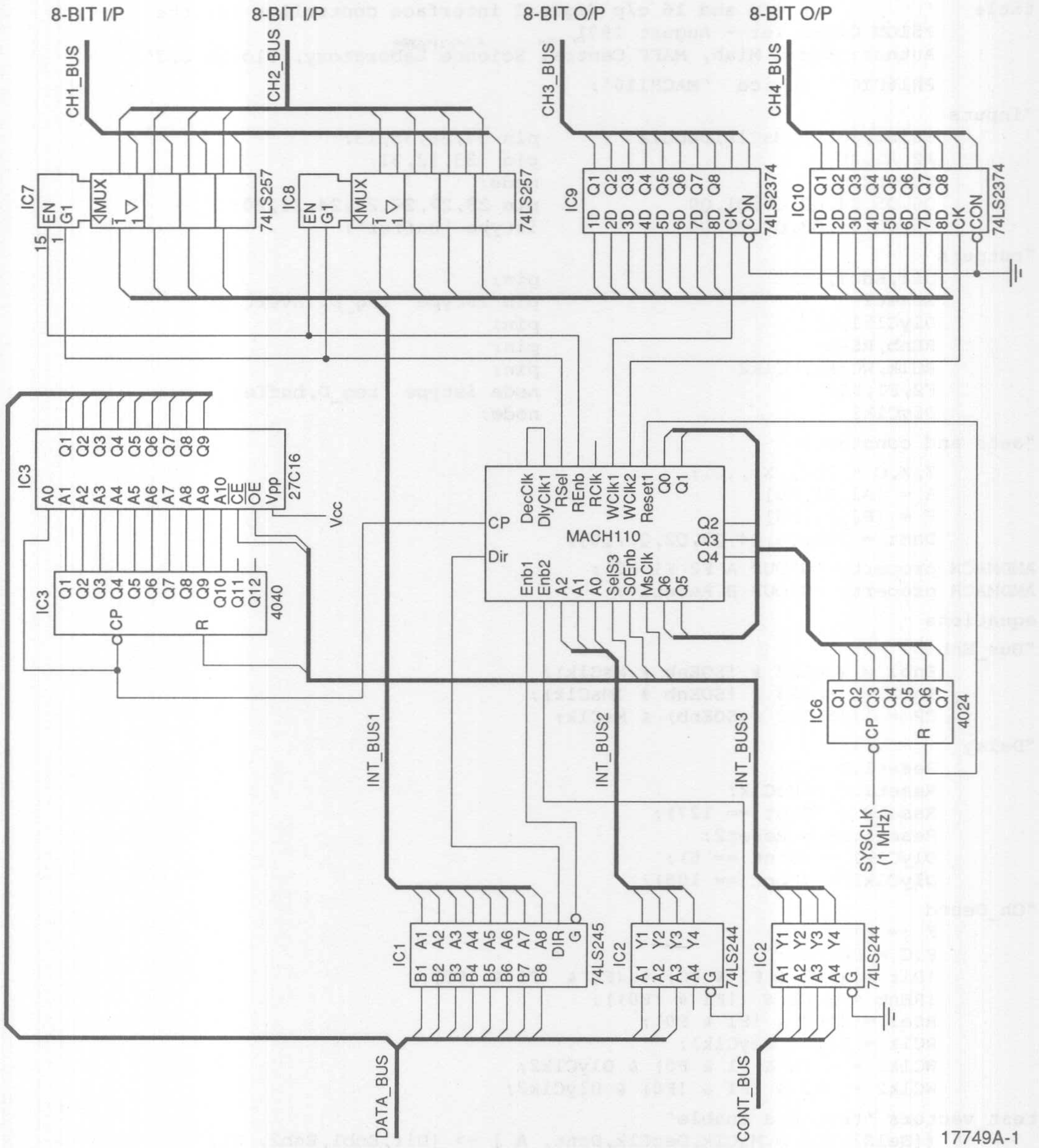


Figure 2-1. Psion Organizer Digital Interface

DESIGN FILE

```
module PN16DIO
title  '       16 i/p and 16 o/p digital interface controller for the
        PSION Organizer - August 1991
        Author: Kamru Miah, MAFF Central Science Laboratory, Slough SL3'
        PN16DIO   device  'MACH110';
"inputs
        SelS3,SOEnb,MsClk,DecClk            pin 37,36,35,13;
        A2,A1,A0                            pin  33,32,31;
        Reset2                              node;
        Q6,Q5,Q4,Q3,Q2,Q1,Q0                pin 28,27,26,25,24,21,20;
        Q6,Q5,Q4,Q3,Q2,Q1,Q0                istype 'buffer';
"outputs
        Dir,Enb1,Enb2                       pin;
        Reset1                              pin istype 'reg_D,invert';
        DlyClk1,CP                          pin;
        REnb,RSel                           pin;
        RClk,WClk1,WClk2                    pin;
        F2,F1,F0                            node istype 'reg_D,buffer';
        DlyClk2                             node;
"sets and constants
        Z,X,C = .Z.,.X.,.C.;
        A = [A2,A1,A0];
        F = [F2,F1,F0];
        Dcnt = [Q6,Q5,Q4,Q3,Q2,Q1,Q0];
AMDMACH property 'GROUP A F2 F1 F0';
AMDMACH property 'GROUP B Reset1';
equations
"Bus_Enb
        Enb1 = (SelS3 # !SOEnb # MsClk);
        Enb2 = (SelS3 # !SOEnb # !MsClk);
        CP = !(!SelS3 & SOEnb) & MsClk;
"Delay
        Reset1.D = 1;
        Reset1.C = MsClk;
        Reset2 = (Dcnt == 127);
        Reset1.AR = Reset2;
        DlyClk1 = (Dcnt == 5);
        DlyClk2 = (Dcnt == 105);
"Ch_Decod
        F := A;
        F.C = DecClk;
        !Dir = (!F2 & F1 & F0) # (F2 & !F1 & !F0);
        !REnb = (!F2 & (!F1 # !F0));
        RSel = (!F2 & !F1 & F0);
        RClk = Dir & DlyClk1;
        WClk1 = (!F2 & F1 & F0) & DlyClk2;
        WClk2 = (F2 & !F1 & !F0) & DlyClk2;
test_vectors 'test Bus Enable'
        ([SelS3,SOEnb,MsClk,DecClk,Dcnt, A ] -> [Dir,Enb1,Enb2, F])
         [  1  ,  X  ,  X  ,  X   ,  X  , X ] -> [ X , 1  ,  1  , X ];
         [  0  ,  1  ,  1  ,  0   ,  5  , 3 ] -> [ X , 1  ,  0  , X ]; "Write Addr
         [  0  ,  1  ,  1  ,  C   ,  5  , 3 ] -> [ 0 , 1  ,  0  , 3 ];
         [  0  ,  1  ,  1  ,  0   ,  5  , 5 ] -> [ X , 1  ,  0  , 3 ];
         [  0  ,  1  ,  1  ,  C   ,  5  , 5 ] -> [ 1 , 1  ,  0  , 5 ];
         [  0  ,  1  ,  1  ,  0   ,  5  , 7 ] -> [ 1 , 1  ,  0  , 5 ];
         [  0  ,  1  ,  1  ,  0   , 10  , 7 ] -> [ 1 , 1  ,  0  , 5 ];
         [  0  ,  1  ,  0  ,  X   ,  X  , X ] -> [ 1 , 0  ,  1  , X ]; "Read Data
```

```
test_vectors 'test Delay Operation'
        ([SelS3,SOEnb,MsClk,Dcnt  ] -> [Reset1,Reset2,DlyClk1,DlyClk2])
        [   1 ,  X  ,  X  ,  X    ] -> [  X   ,  X   ,  X   ,   X  ];
        [   0 ,  1  ,  0  , 127   ] -> [  1   ,  1   ,  0   ,   0  ];"Reset
Counter
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  1    ] -> [  0   ,  0   ,  0   ,   0  ];"Start
Counter
        [   0 ,  1  ,  0  ,  2    ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  3    ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  4    ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  5    ] -> [  0   ,  0   ,  1   ,   0  ];"Generate 1st
        [   0 ,  1  ,  0  ,  6    ] -> [  0   ,  0   ,  0   ,   0  ];"delay pulse
        [   0 ,  1  ,  0  ,  7    ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  , 103   ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  , 104   ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  , 105   ] -> [  0   ,  0   ,  0   ,   1  ];"Generate 2nd
        [   0 ,  1  ,  0  , 106   ] -> [  0   ,  0   ,  0   ,   0  ];"delay pulse
        [   0 ,  1  ,  0  , 107   ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  , 125   ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  , 126   ] -> [  0   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  , 127   ] -> [  1   ,  1   ,  0   ,   0  ];"Stop Counter
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
        [   0 ,  1  ,  0  ,  0    ] -> [  1   ,  0   ,  0   ,   0  ];
test_vectors 'test R/W Operation'
        ([DecClk,Dcnt,DlyClk2, A ]->[REnb,RSel,WClk1,WClk2])
        [  X   ,  X ,   X   ,  X ]->[ X  , X  ,  X  ,  X  ];
        [  C   ,  5 ,   0   ,  0 ]->[ 0  , 0  ,  0  ,  0  ];"Idle State
        [  0   ,  X ,   0   ,  1 ]->[ 0  , 0  ,  0  ,  0  ];
        [  C   ,  5 ,   0   ,  1 ]->[ 0  , 1  ,  0  ,  0  ];"Select CH1 (R)
        [  0   ,  X ,   0   ,  1 ]->[ 0  , 1  ,  0  ,  0  ];
        [  C   ,  5 ,   0   ,  2 ]->[ 0  , 0  ,  0  ,  0  ];"Select CH2 (R)
        [  0   ,  X ,   0   ,  2 ]->[ 0  , X  ,  0  ,  0  ];
        [  C   ,  5 ,   0   ,  3 ]->[ 1  , X  ,  0  ,  0  ];"Select CH3 (W)
        [  0   ,  X ,   0   ,  3 ]->[ 1  , X  ,  0  ,  0  ];
        [  C   , 105,   1   ,  3 ]->[ 1  , X  ,  X  ,  0  ];"Latch o/p DATA
        [  0   ,  X ,   1   ,  3 ]->[ 1  , X  ,  X  ,  0  ];
        [  C   ,  5 ,   0   ,  4 ]->[ 1  , X  ,  0  ,  0  ];"Select CH4 (W)
        [  0   ,  X ,   0   ,  4 ]->[ 1  , X  ,  0  ,  X  ];
        [  C   , 105,   1   ,  4 ]->[ 1  , X  ,  0  ,  X  ];"Latch o/p DATA
        [  0   ,  X ,   0   ,  0 ]->[ 1  , X  ,  0  ,  0  ];
        [  C   ,  5 ,   0   ,  0 ]->[ 0  , X  ,  0  ,  0  ];"Idle state
end
```

FITTER FILE

Note: This file has been condensed in order to save trees.

```
AMD MACH FITR - MARKET RELEASE (1-24-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990
Flags Used:             Unplace=False           Max Packing=True
Flags Used:          Expand Small=False          Expand All=False
************************************************************************************
     Mach PLD Fitter - v 1.46             16 i/p and 16 o/p digital interface
                                          controller for the psion organizer
************************************************************************************

*** Timing Analysis for Signals
 Parameter   Min   Max          Signal List (Those having Max delay.)
      Tpd     1     2              RClk           WClk1           WClk2
      Tsu     1     2            Reset1
      Tco     0     2              RClk

 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Register clock to combinatorial output
 Tcr - Register thru combinatorial logic to setup
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                 Available         Used         Remaining
         Clocks:     2               2              0
           Pins:    38              25             13    ->    65%
      I/O Macro:    32              11             21
    Total Macro:    32              16             16
 Product Terms:    128              19             64    ->    50%

MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
*** Last Equations Placed in Blocks
Weakly -
Assign -            RClk
*** Block Partitioning Results
                 Array    Macros      # I/O    Buried   Product     Signal
                Inputs    Remain      Macro    Logic     Terms      Fanout
   Block-> A      13        7           4        5        36          6
   Block-> B       7        9           7        0        28          1
*** Block Signal List
Block-> A                 CP             Enb2             Enb1          DlyClk2
                      Reset2          DlyClk1               F0               F1
                          F2
Block-> B               RClk             RSel             REnb              Dir
                       WClk2            WClk1           Reset1
```

```
*** Feedback Map -         16 i/p and 16 o/p digital interface controller for the
 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|       A0 : 0|     |21| A2          DlyClk1 : 0|     |21|
         | 1|          | 1|     |20| A1                  | 1|     |20|
         | 2|          | 2|     |19|                /Dir : 2|     |19|
         | 3|       Q6 : 3|     |18| MsClk               | 3|     |18|
         | 4|       Q5 : 4|     |17|                  F0 : 4|     |17|
         | 5|       Q4 : 5|     |16|                  F1 : 5|     |16|
         '--'       Q3 : 6|     |15: SOEnb            F2 : 6|     |15|
                    Q2 : 7|     |14: SelS3        Reset2 : 7|     |14|
                       | 8|     |13: Q0          DlyClk2 : 8|     |13|
                       | 9|     |12: Q1                  | 9|     |12|
                       |10|     |11|                     |10|     |11|
                       '--+-u--u+--'                     '--+-u--u+--'
*** Logic Map -         16 i/p and 16 o/p digital interface controller for the
 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|    /Enb1 | 0| 1   |21|                     | 0| .   |21|
         | 1|    /Enb2 | 1| 1   |20|                     | 1| .   |20|
   DecClk| 2|       CP | 2| 2   |19|                     | 2| .   |19|
       A1| 3|  DlyClk1 | 3| 1   |18|                     | 3| .   |18|
       A2| 4|       F0 | 4| 1   |17|                     | 4| .   |17|
    MsClk| 5|       F1 | 5| 1   |16|                /Dir | 5| 2   |16|
         '--'       F2 | 6| 1  .|15|               WClk1 | 6| 1  .|15|
                Reset2 | 7| 1  .|14|                     | 7| .  1|14| /Reset1
               DlyClk2 | 8| 1  .|13|                     | 8| .  1|13| RSel
                       | 9| .  .|12|                     | 9| .  1|12| RClk
                       |10| .  .|11|               WClk2 |10| 1  2|11| /REnb
                       '--+-u--u+--'                     '--+-u--u+--'
```

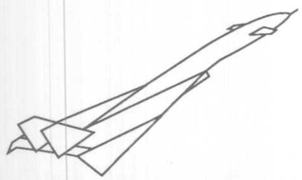
```
*** Pin Map -         16 i/p and 16 o/p digital interface controller for the




                                  /Enb1
                               /Enb2  |        .
                               CP  |  |        |  /Reset1
                       DlyClk1  |  |  |        |  |  RSel
                          .  |  |  |  |        |  |  |  RClk
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
                    | 7                                   39|/REnb
                    | 8                  G  V             38|WClk2
                    | 9                  n  c             37|SelS3
                    |10                  d  c             36|SOEnb
                    |11                                   35|MsClk
                Gnd |12            MACH-110               34| Gnd
              DecClk|13                                   33|A2
                    |14             V  G                  32|A1
                    |15             c  n                  31|A0
                    |16             c  d                  30|WClk1
                    |17                                   29|/Dir
                    |   1  1  2  2  2  2  2  2  2  2  2     |
                    |   8  9  0  1  2  3  4  5  6  7  8     |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                        '  |  |  |        |  |  |  |  Q6
                           '  |  |        |  |  |  Q5
                             Q0  |        |  |  Q4
                                Q1        |  Q3
                                          Q2
The Design Doc is  stored in ===> pn16dio.Rpt
The Jedec Data is  stored in ===> pn16dio.Jed
The Placements are stored in ===> pn16dio.Plc
The  Fuse Plot is  stored in ===> pn16dio.Xpt

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: pn16dio
```

PLACEMENT FILE

```
; Flags Used:           Unplace=False         Max Packing=True
; Flags Used:      Expand Small=False          Expand All=False
; FITR generated placements
Pin    37           SelS3  ; Inp          ; B   9
Pin    36           SOEnb  ; Inp          ; B   8
Pin    35           MsClk  ; Inp          ; I   5
Pin    13          DecClk  ; Inp          ; I   2
Pin    33              A2  ; Inp          ; I   4
Pin    32              A1  ; Inp          ; I   3
Pin    31              A0  ; Inp          ; B   7
Pin    28              Q6  ; Inp          ; B   4
Pin    27              Q5  ; Inp          ; B   3
Pin    26              Q4  ; Inp          ; B   2
Pin    25              Q3  ; Inp          ; B   1
Pin    24              Q2  ; Inp          ; B   0
Pin    21              Q1  ; Inp          ; A  15
Pin    20              Q0  ; Inp          ; A  14
Pin    29            /Dir    Comb         ; B   5
Pin     2           /Enb1    Comb         ; A   0
Pin     3           /Enb2    Comb         ; A   1
Pin    42         /Reset1     Reg         ; B  14
Pin     5         DlyClk1    Comb         ; A   3
Pin     4              CP    Comb         ; A   2
Pin    39           /REnb    Comb         ; B  11
Pin    41            RSel    Comb         ; B  13
Pin    40            RClk    Comb         ; B  12
Pin    30           WClk1    Comb         ; B   6
Pin    38           WClk2    Comb         ; B  10
Node    9          Reset2    Comb         ; A   7
Node    8              F2     Reg         ; A   6
Node    7              F1     Reg         ; A   5
Node    6              F0     Reg         ; A   4
Node   10         DlyClk2    Comb         ; A   8
; Group Mach_Seg_A CP Enb2 Enb1 DlyClk2 Reset2 DlyClk1 F0 F1 F2
; Group Mach_Seg_B RClk RSel REnb Dir WClk2 WClk1 Reset1
```

References

1. Data I/O Corporation, *ABEL Design Softuare User Manual,* (1990)
2. Advanced Micro Devices, *MACH Devices—High Density EE CMOS Programmable Logic1991 Data Book,* (1991)
3. Texas Instruments, *TTL Data Book*, Vol. 1, (1987)



Profibus Repeater: MACH110

Advanced Micro Devices

Application Note

by Uwe Nolte, Germany

POSING THE PROBLEM

A circuit is to be developed that can regenerate the serial data according to the Profibus Standard. The following requirements are to be noted:

- Two connectable bus segments
- Maximum three components (1*MACH110, 2*driver per EIA485 Standard)
- Repeater transit time maximum of one bit length
- Digital filters for both bus segments
- Independent direction recognition and switching
- Sensing of received data in the middle of the bit
- Chronological regeneration of bit lengths
- Continuous transmission of several bytes
- Generation of all times from one master clock
- Oscillator and Reset logic integrated in the PLA

UTILIZATION OF MACH110 FEATURES

- Uses two timing inputs (dynamic clock generation occurs on-chip)
- 30 of 32 product term groups used
- Asynchronous reset used by blocks
- Outputs combined and clocked and used buffered and inverted
- Uses "high Z"-function of the outputs

Table 1. Fitter Resources

	Available	Used	Remaining
Clocks:	2	2	0
Pins:	38	32	6 –> 84%
I/O Macro:	32	26	6
Total Macro:	32	26	6
Product Terms:	128	76	8 –> 92%

BRIEF EXPLANATION OF THE PROFIBUS

The Profibus is a format described in DIN V29245 T1 and T2 for serial communication of several subscribers and builds on the EIA 485 method. The EIA 485 is essentially a closed, bidirectional, two-wire line to which up to 32 subscribers can be connected via short taps. Now where the number is not sufficient, several of these segments can be circuited together via repeaters. They decouple both segments and regenerate the transmitted signals. For the repeaters described below, essentially three points are important:

- Every transferred byte consists of 11 bits: Start bit, 8 data bits, parity bit and stop bit
- Between two transferred bytes, no pause may be made, i.e. for each stop bit, the next start bit must follow immediately
- Between two transmissions at least 33 idle bit times must occur so that the connected subscribers can be resynchronized.

DESCRIPTION OF THE EXISTING REPEATER DESIGN

Brief Overview

The design consists essentially of three units:

- Generation of a reset signal
- Provision of the fixed clock pulses
- Repeater logic

For the production of the reset signal and of the clock pulses, we need only two combinatorial equations, besides a few hardware circuits; however, the repeater logic consists of five state machines that are operated with two differing clock pulses, two 3-bit counters with variable preload, and also five additional equations to generate the output signals. With the five state machines the functional unit of digital filters A and B, the preselection, the bit regeneration and also the timer control can be implemented. The variable clock pulse for bit regeneration and the output signals is obtained dynamically from the timer control and the two counters. The other output signals are used to drive the EIA485 driver from the existing state machines. A detailed description of the functional units is provided below:

Generation of the Reset Signal

To generate the Reset signal, a Schmitt trigger function is needed to avoid oscillation of the circuit. A buffer is provided for this that produces the necessary positive feedback. See Figure 3-1.

When turned off, the capacitor is discharged. After turn-on, the capacitor slowly charges up via R1. As soon as the input IRES reaches its switching threshold, the attendant output IRES_O goes low-ohm and quickly charges up the capacitor. Resistor R2 limits the charge

Publication# 17751 Rev. A Amendment/0
Issue Date: May 1993

current for C1 and decouples output and input, so that an external signal can be injected to IRES.

Generation of Clock Pulses

An inverter was provided to ensure the necessary clock pulses. Now it is very easy to construct a quartz oscillator. See Figure 3-2.

The signal applied to OSZ_I will be output (negated due to the inverter function) to OSZ_O and fed back via R1, that is, R1 is used to initiate the oscillation. Since in case of resonance, the apparent resistance of Q1 is clearly less than R1, the resulting frequency will be determined by Q1.

The time for transient oscillation of the quartz oscillator is inversely proportional to the bandwidth; several milliseconds may be needed here. During this time, the Reset signal must be applied in order not to allow any undefined behavior of the circuit.

Hardware Requirements

The following signals are needed for connection of the EIA485 driver:

- RXD_A: Received data on channel A (receiver is always active)
- RXD_B: Received data on channel B (receiver is always active)
- EN_A: Activation of sender on channel A (high-active)
- EN_B: Activation of sender on channel B (high-active)
- TXD: Sender data (signal applied to both drivers)

On the MACH110 the following signals are to be connected to each other:

- CLK_VARO –> CLK_VAR : dynamic clock generator for CLK2
- OSZ_0 –> CLK_FIX : clock pulse

Data Flow Plan for Repeater Logic

The repeater logic is organized into several functional units; these are mostly designed as state machines.

Function of the Digital Filter

The digital filters for channels A and B are of the same configuration and have a filter depth of two, that is, an input signal is picked up as starting value only after it has been twice sensed with the same result. The following status bits are used for this:

Channel A: RXD_FA and CHA_1
Channel B: RXD_FB and CHB_1

The status coding is chosen so that bits RXD_FA and RXD_FB correspond directly to the filtered and inverted input signal.

Functioning of the Preselection

The preselection consists of the status bits VS_ACTV—denotes the data reception and SEL_AB—selects the data channel.

In the inactive state, both bits are reset and, upon reception of one start bit, an active-state (VS_AACT or VS_BACT) will be assumed and retained. Due to an early recognition of the transition conditions of the digital filter, the active states will be assumed in the same timing pulse in which RXD_FA or RXD_FB assumes these states. In this case, channel A has priority over B. Return to the neutral state is only possible via the asynchronous Reset. This Reset can only be executed blockwise by the MACH110; therefore the status bits need not be of inverted design, and lie in the same block with the status bits of the bit regeneration and timer control.

Function of the Bit Regeneration

The state machine of the bit regeneration is constructed of bits REG_3, REG_2, REG_1 and REG_0, except for the status coding, they have no additional significance. The timing of these bits takes place due to the dynamic generated clock pulse and thus makes possible the central sensing and time regeneration. For the bit regeneration, 14 states are defined, one each for start, data, parity and stop bit, and in addition, neutral, first-time delay and reset status will be defined. The delay status (REG_DLY) is needed once at the beginning of each transmission, in order to bridge the time between received start flank and sensing timepoint. The reset state is used as trigger for the asynchronous Reset, that also acts on the current status bit. The last two, still possible states REG_AGN and REG_UDF, will not be used and lead directly to the neutral state.

Function of the Timer Controller and of Prescalars A and B

The following status bits are used:

TCO_2: Status bit for timer control, no particular significance

TCO_1: Status bit for timer control, simultaneously start bit, prescalar B

TCO_0: Status bit for timer control, simultaneously start bit, prescalar A

TAQ_2, TAQ_1, TAQ_0: Counter bits, prescalar A

TAQ_2, TBQ_1, TBQ_0: Counter bits, prescalar B.

Prescalars A and B are implemented as counters with variable preload potential in equation form and are started and stopped by the state machine "timer controller." They form the basis for the dynamic generated clock pulse, since they reduce the fixed clock frequency by a factor of eight to the necessary baud rate. The preload potential is needed to delay the sensing timepoints compared to the start flank by one half bit. For the first start bit, the prescalar A must be delayed by (n/2 – 2) clock pulses, since just one clock pulse was "consumed" by the bit regeneration and another one is "consumed" by the timer control. In all the following Start bits, only the timer control "consumes" one clock pulse, so that here the prescalars have to be delayed by only (n/2 – 1) clock pulses.

In the timer control there are eight possible states, only five of which are used. The other three lead directly to the neutral state. The possible states of the timer control are as follows:

TCO_Z00:	Neutral state, both prescalars stopped
TCO_Z01:	Prescalar A is running, B is stopped
TCO_ZO2:	Prescalar A is still running, B has already started
TCO_Z03:	Prescalar A is stopped, only B is running
TCO_Z04:	Prescalar A was started, prescalar B is still running

The intersection of the Start/Stop conditions of the prescalar is needed to achieve synchronization of the prescalars with the new start flank in case of on-going transmission of the Stop bit.

Generation of the Output Signals

The following output signals are generated from the state machines and prescalars:

CLK_VARO:	dynamic generated clock pulse for input CLK_VAR
EN_A:	activates driver to bus segment A (high active)
EN_B:	activates driver to bus segment B (high active)
TXD:	sender data for both bus segments
DTR:	auxiliary signal to generate EN_A and EN_B (data transmit)

The auxiliary signal DTR was introduced to minimize the number of product terms and to improve the legibility of the logic. It is set for each bit during 11 transferred bits and is derived from the bits of bit regeneration.

EN_A and EN_B are derived combinatorially from this auxiliary signal and the status bits for preselection.

TXD however, is derived only from the bits of the prescalar and digital filter and is powered as the clocked output from the variable clock pulse. The output CLK_VARO belongs thematically to the timer control block, but is discussed here as an output signal. This output must be connected on the hardware side with the input pin CLK_VAR, since here the sensing pulse is generated dynamically. In an inactive bit regeneration, the four-fold bit pulse is available here, otherwise the bit clock pulse for central sensing. The basis for this is the status bits of the bit regenerator and the timer controller and also the prescalar bits.

Profibus-Repeater: MACH110

Advanced Micro Devices

Application Note

by Uwe Nolte, Germany

AUFGABENSTELLUNG

Es ist eine Schaltung zu entwickeln, die serielle Daten nach der Profibus-Norm regenerieren kann. Folgende Anforderungen sind dabei zu beachten:

- Zwei Bussegmente anschliessbar
- Max drei Bausteine (1*MACH110, 2*Treiber nach EIA485-Norm)
- Repeaterdurchlaufzeit max eine Bitlänge
- Digitale Filter für beide Bussegmente
- Selbsttätige Richtungserkennung und – umschaltung
- Abtastung der empfangenen Daten in Bitmitte
- Zeitliche Regeneration der Bitlängen
- Lückenlose Übertragung mehrerer Bytes
- Generierung aller Zeiten aus einem Muttertakt
- Oszillator– und Resetlogik im PLA integrieren

AUSNUTZUNG DER MACH110—FEATURES

- Zwei Takteingänge genutzt (es erfolgt dyn. Takterzeugung onChip)
- 30 von 32 Produkttermgruppen genutzt
- Asynchronen Reset blockweise genutzt
- Ausgänge komb. und getaktet sowie puffernd und invertierend genutzt
- "High Z" – Funktion der Ausgänge genutzt

Table 1. Fitter Resources

	Available	Used	Remaining
Clocks:	2	2	0
Pins:	38	32	6 –> 84%
I/O Macro:	32	26	6
Total Macro:	32	26	6
Product Terms:	128	76	8 –> 92%

KURZERLÄUTERUNG ZUM PROFIBUS

Der Profibus ist ein in DIN V29245 T1 und 2 beschriebenes Format zur seriellen Kommunikation mehrerer Teilnehmer und baut auf der EIA 485 – Physik auf. Die EIA 485 ist im wesentlichen eine abgeschlossene bidirektionale Zweidrahtleitung, an der bis zu 32 Teilnehmer über kurze Abgriffe anschliessbar sind. Sofern die Anzahl nicht ausreicht, können mehrere dieser Segmente über Repeater zusammen-geschaltet werden. Dieser entkoppelt beide Segmente und regeneriert die übertragenen Signale. Für den nachfolgend beschriebenen Repeater sind im wesentlichen drei Punkte wichtig:

- Jedes übertragene Byte besteht aus 11Bits: Startbit, 8Datenbits, Paritätsbit und Stopbit.
- Zwischen zwei übertragene Bytes darf keine Pause gemacht werden, d.h. auf jedes Stopbit muss sofort das nächste Startbit folgen.
- Zwischen zwei übertragungen muss mindestens 33Bitzeiten Ruhe herrschen, damit sich die angeschlossenen Teilnehmer neu synchronisieren können.

BESCHREIBUNG DES VORLIEGENDEN REPEATERDESIGNS

Kurzübersicht

Das Design besteht im wesentlichen aus drei Blöcken:

- Erzeugung eines Resetsignals
- Bereitstellung der festen Taktversorgung
- Repeaterlogik

Für die Erzeugung des Resetsignals und der Taktversorgung sind ausser einiger Hardwarebeschaltung nur zwei kombinatorische Gleichungen nötig, die Repeater– logik hingegen besteht aus 5 Zustandsmaschinen die mit zwei unterschiedlichen Takten betrieben werden, zwei 3Bit–Zählern mit variablem Preload sowie 5 weiteren Gleichungen zur Erzeugung der Ausgangssignale. Mit den fünf Zustandsmaschinen sind die Funktionsblöcke digitale Filter A und B, die Vorselektion, die Bitregenerierung sowie die Timersteuerung realisiert. Der variabler Takt für die Bitregenerierung und die Ausgangssignale wird dynamisch aus der Timersteuerung und den beiden Zählern gewonnen. Die übrigen Ausgangssignale dienen der Ansteuerung der EIA485–Treiber aus den vorhandenen Zustandsmaschinen. Nachfolgend eine detaillierte Beschreibung der Funktionsblöcke:

Erzeugung des Reset-Signals

Zur Erzeugung des Reset-Signals ist eine Schmitt–Triggerfunktion notwendig, um ein Oszillieren der Schaltung zu vermeiden. Hierfür ist ein Buffer vorhanden, der die notwendige Mitkopplung erzeugt (Figure 3-1).

Im ausgeschalteten Zustand ist der Kondensator entladen. Nach dem Einschalten lädt sich der Kondensator über R1 langsam auf. Sobald der Eingang IRES seine Schaltschwelle erreicht, wird der zugehörige Ausgang IRES_O niederohmig und lädt den Kondensator schnell auf. Der Widerstand R2 begrenzt den Ladestrom für C1 und entkoppelt Ausgang und Eingang, so dass an auch ein externes Signal eingespeist werden kann.

Bereitstellen der Taktversorgung

Um die notwendige Taktversorgung zu gewährleisten, wurde ein Inverter vorgesehen. Hiermit lässt sich sehr einfach ein Quarz–Oszillator aufbauen (Figure 3-2).

Das an OSZ_I anliegende Signal wird wegen der Inverterfunktion an OSZ_O negiert ausgegeben und über R1 wieder leicht zurückgekoppelt, d.h. R1 dient dem Anstossen der Oszillation. Da im Resonanzfall der Schein–widerstand von Q1 deutlich kleiner als R1 ist, wird die resultierende Frequenz durch Q1 bestimmt. Die Zeit zum Einschwingen des Quarzoszillators ist umgekehrt proportinal zur Bandbreite, hier können mehrere Millisekunden erreicht werden. Während dieser Zeit muss das Reset–Signal anliegen, um kein undefiniertes Verhalten der Schaltung zuzulassen.

Hardware–Anforderungen

Für den Anschluss der EIA485–Treiber sind folgende Signale vorgesehen:

- RXD_A: Empfangene Daten auf Kanal A (Empfaenger ist immer aktiv)
- RXD_B: Empfangene Daten auf Kanal B (Empfaenger ist immer aktiv)
- EN_A: Aktivierung Sender auf Kanal A (high–aktiv)
- EN_B: Aktivierung Sender auf Kanal B (high–aktiv)
- TXD: Sendedaten (Signal liegt an beiden Treibern an)

Am MACH110 sind folgende Signale miteinander zu verbinden:

- CLK_VARO –> CLK_VAR: dynamische Takterzeugung für CLK2
- OSZ_O –> CLK_FIX: Taktversorgung

Datenflussplan zur Repeaterlogik

Die Repeaterlogik gliedert sich in mehrere funktionale Blöcke, diese sind meistens als State-Machines ausgebildet.

Funktionsweise der digitalen Filter

Die digitalen Filter für die Kanäle A und B sind gleich aufgebaut und besitzen eine Filtertiefe von Zwei, d.h. dass ein Eingangssignal erst bei zweimaliger gleicher Abtastung als Ausgangswert übernommer wird. Folgende Zustandsbits werden hierfür benutzt:

Kanal A: RXD_FA und CHA_1
Kanal B: RXD_FB und CHB_1.

Die Zustandskodierung ist so gewählt, dass die Bits RXD_FA und RXD_FB direkt dem gefilterten und invertiertem Eingangssignal entspricht.

Funktionsweise der Vorselektion

Die Vorselektion besteht aus den Zustandsbits VS_ACTV—zeigt den Datenempfang an und SEL_AB—selektiert den Datenkanal.

Im inaktiven Zustand sind beide Bits zurückgesetzt und bei Empfang eines Startbits wird ein Aktiv–Zustand (VS_AACT oder VS_BACT) eingenommen und beibehalten. Durch ein frühzeitiges Erkennen der Übergangsbedingungen der digitalen Filter werden die Aktiv–Zustände im gleichen Takt eingenommen, in dem RXD_FA oder RXD_FB diesen einnehmen. Hierbei hat Kanal A Priorität vor B. Die Rückkehr in den neutralen Zustand ist nur über den asynchronen Reset möglich. Dieser kann beim MACH110 nur blockweise ausgeführt werden, daher müssen die Zustandsbits nicht invertiert ausgeführt werden und mit den Zustandsbits der Bitregenerierung und Timersteuerung im gleichen Block liegen.

Funktionsweise der Bitregenerierung

Die Zustandsmaschine der Bitregenerierung ist aus den Bits REG_3, REG_2, REG_1 und REG_0 aufgebaut, diese haben ausser der Zustandskodierung keine weitere Bedeutung. Die Taktung dieser Bits erfolgt durch den dynamisch erzeugten Takt und ermöglicht somit die Mittenabtastung und zeitliche Regeneration. Für die Bitregenerierung werden 14 Zustände definiert, je einer für Start–,Daten–, Paritäts– und Stopbit, ausserdem werden noch Neutral, erstmalige Verzögerung und Rücksetzzustand definiert. Der Verzögerungszustand (REG_DLY) ist bei Beginn jeder Übertragung einmal nötig, um die Zeit zwischen empfangener Startflanke und Abtastzeitpunkt zu Überbrücken. Der Rücksetz–zustand dient als Auslöser für den asynchronen Reset, der auch auf die hiesigen Zustandsbits wirkt. Die beiden letzten noch möglichen Zustände REG_AGN und REG_UDF werden nicht verwendet und führen direkt zum neutralen Zustand.

Funktionsweise der Timersteuerung sowie der Vorteiler A und B

Hier werden folgende Zustandsbits verwand:

TCO_2: Zustandsbit Timersteuerung ohne besondere Bedeutung

TCO_1: Zustandsbit Timersteuerung gleichzeitig Startbit Vorteiler B

TCO_0: Zustandsbit Timersteuerung gleichzeitig Startbit Vorteiler A

TAQ_2, TAQ_1, TAQ_0: Zählerbits Vorteiler A

TBQ_2, TBQ_1, TBQ_0: Zählerbits Vorteiler B.

Die Vorteiler A und B sind als Zähler mit variabler Preload-Möglichkeit in Gleichungsform realisiert und werden durch die Zustandsmaschine 'Timersteuerung' gestartet und gestoppt. Sie bilden die Basis für den dynamisch erzeugten Takt, indem sie die feste Taktfrequenz um Faktor 8 auf die benötigte Baudrate erringern. Die Preloadmöglichkeit ist nötig, um die Abtastzeitpunkte gegeüber der Startflanke um ein halbes Bit zu verzögern. Beim 1. Startbit muss Vorteiler A (n/2 – 2) Takte verzögern, da bereits ein Takt von der Bitregenerierung und ein weiterer von der Timer–steuerung 'verbraucht' wurde. Bei allen folgenden Startbits 'verbraucht' nur die Timersteuerung einen Takt, so dass hier die Vorteiler nur um (n/2 – 1) Takt verzögern müssen.

Bei der Timersteuerung gibt es 8 mögliche Zustände, von denen nur 5 genutzt werden. Die übrigen drei führen direkt zum neutralen Zustand. Die möglichen Zustände der Timersteuerung lauten:

TCO_Z00: neutraler Zustand, beide Vorteiler gestoppt

TCO_Z01: Vorteiler A läuft, B ist gestoppt

TCO_Z02: Vorteiler A läuft noch, B ist schon gestartet

TCO_Z03: Vorteiler A ist gestoppt, nur B läuft

TCO_Z04: Vorteiler A wurde gestartet, Vorteiler B läuft noch

Die Überschneidung der Start/Stop-Bedingungen der Vorteiler ist nötig, um bei laufender Übertragung des Stopbits wieder die Synchronisation der Vorteiler mit der neuen Startflanke zu erreichen.

Generierung der Ausgangssignale

Folgende Ausgangssignale werden aus den Zustandsmaschinen und Vorteilern generiert:

CLK_VARO: dynamisch erzeugter Takt für Eingang CLK_VAR

EN_A: aktiviert Treiber auf Bussegment A (high aktiv)

EN_B: aktiviert Treiber auf Bussegment B (high aktiv)

TXD: Sendedaten für beide Bussegmente

DTR: Hilfssignal zur Generierung von EN_A und EN_B (Data TRansmit)

Das Hilfssignal DTR wurde eingeführt, um die Anzahl der Produktterme zu minimieren und die Lesbarkeit der Logik zu verbessern. Es ist während der elf zu übertragenen Bits je Byte gesetzt und leitet sich aus den Bits der Bitregenerierung her.

EN_A und EN_B werden aus diesem Hilfssignal und den Zustandsbits der Vorselektion kombinatorisch hergeleitet.

TXD leitet sich hingegen nur aus den Bits der Vorsteuerung und digitalen Filter ab und wird als getakteter Ausgang vom variablen Takt versorgt. Der Ausgang CLK_VARO gehört thematisch zum Block Timersteuerung, wird aber als Ausgangssignal hier besprochen. Dieser Ausgang muss hardwaremässig mit dem Eingangspin CLK_VAR verbunden werden, da hier der Abtasttakt dynamisch erzeugt wird. Bei inaktiver Bitregenerierung ist hier der vierfache Bittakt verfügbar, sonst der Bittakt für mittige Abtastung. Basis hierfür sind die Zustandsbits der Bitregenerierung und der Timersteuerung sowie die Vorteilerbits.

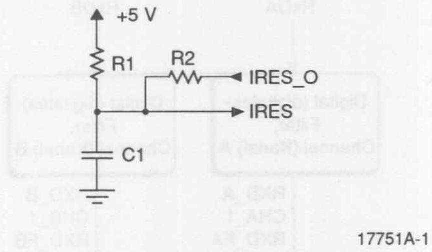


Figure 3-1. Generation of the Reset Signal

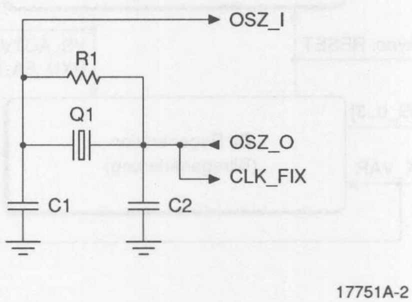


Figure 3-2. Generation of Clock Pulses

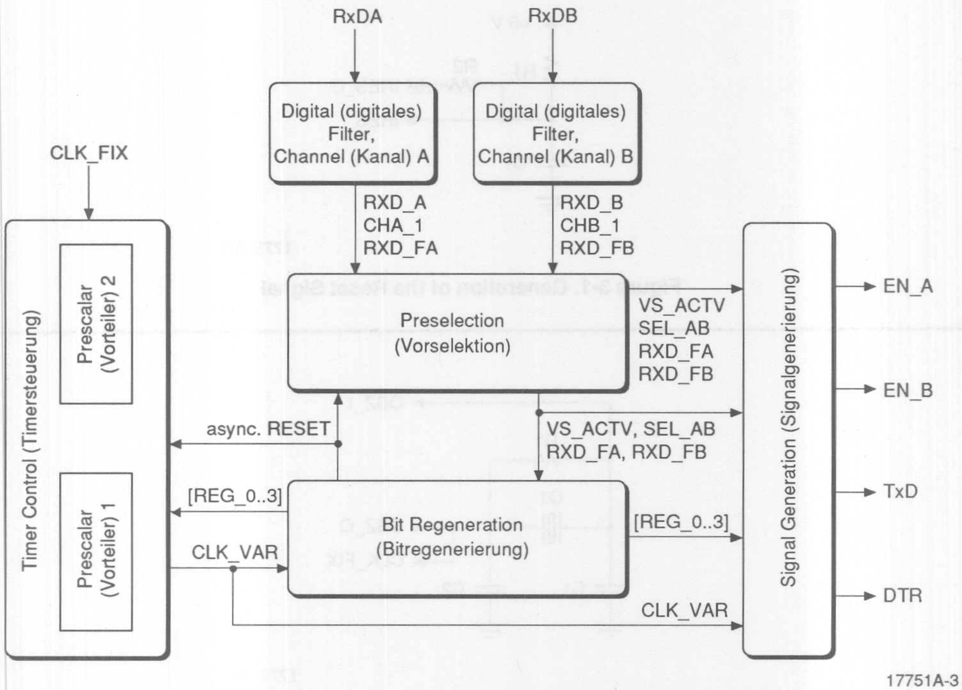
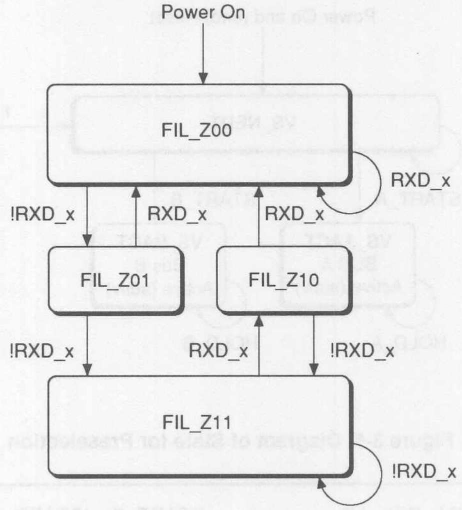


Figure 3-3. Data Flow Plan for the Profibus Repeater

Adaptation to various baud rates is possible via the variation of the clock frequency CLK_FIX; this is always a factor of 8 greater than the baud rate.

Key: 1–digital filter, channel A 2–digital filter, channel B 3–prescalar, 4–timer control, 5–preselection, 6-bit regeneration, 7–signal generation.

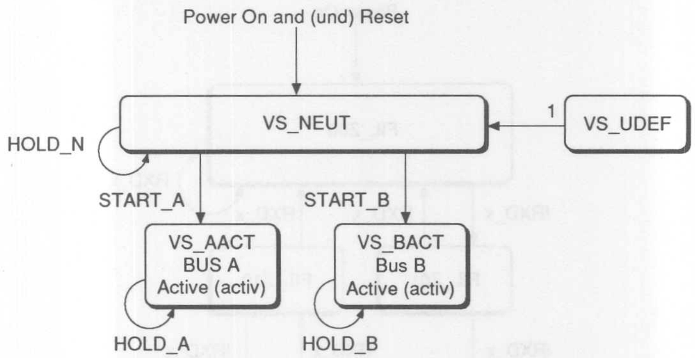


17751A-4

Figure 3-4. Diagram of State of the Digital Filters

Coding of states:	RXD_Fx	CHx_1
FIL_Z00	0	0
FIL_Z01	0	1
FIL_Z10	1	0
FIL_Z11	1	1

Note: Due to the above selection of the status bits, RXD_Fx corresponds to the inverted and filtered input signal RXD_x.



17751A-5

Figure 3-5. Diagram of State for Preselection

Status coding:	VS_ACTV	SEL_AB
VORS_Z_NEUT	0	0
VORS_Z_AAC	1	1
VORS_Z_BACT	1	0
VORS_Z_UDEF	1	1

Transition conditions:
START_A = (FIL_A_VERGL = = FIL_Z01) & IRXD_A # RXD_FA.fb

START_B = !START_A & (FIL_B_VERGL = = FIL_Z01) & !RXD_B # RXD_FB.fb
HOLD_N = !START_A & !START_B

HOLD_A = 1
HOLD_B = 1

Return from VS_AACT and VS_BACT to VS_NEUT takes place via the asynchronous reset.

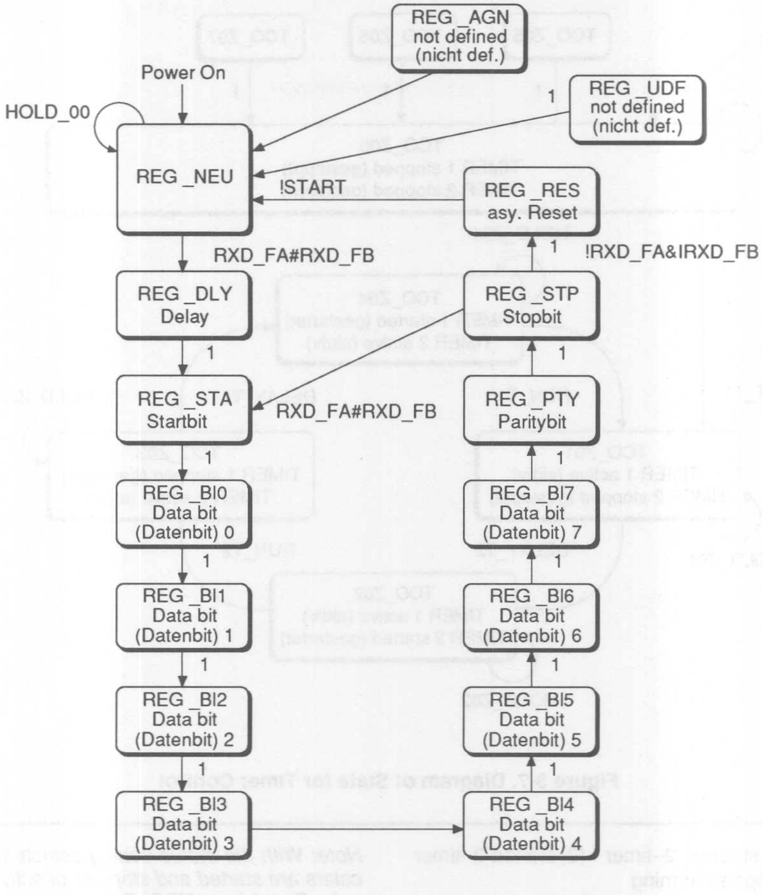


Figure 3-6. Diagram of State for Bit Regeneration

Key: 1–not defined

Key: 1–status coding (per gray code)

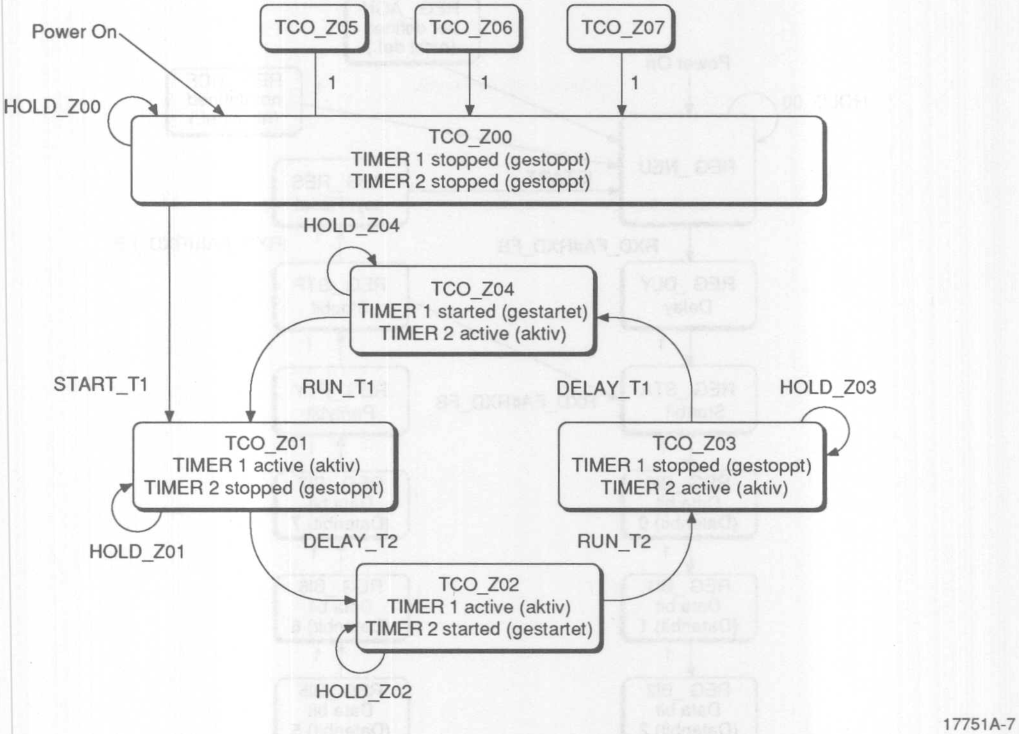


Figure 3-7. Diagram of State for Timer Control

Key: 1–timer 1 (2) stopped 2–timer 1 (2) started 3–timer 2 (1) active 4–stopped/running

Status coding:	TCO_2	TCO_1	TCO_0
TCO_Z00	0	0	0
TCO_Z01	0	0	1
TCO_Z02	0	1	1
TCO_Z03	0	1	0
TCO_Z04	1	1	1
TCO_Z05	1	1	0
TCO_Z06	1	0	1
TCO_Z07	1	0	0

Note: With the status coding selected here, the prescalars are started and stopped directly via bits TCO_0 ...1. Reset from TCO_Z00..04 occurs via the async. Reset

DESIGN FILE

```
" Abel4 - Einstellungen
"=======================
"           Compile Options:    No Listing
"           Reduction Options:  Reduce by Pin, Fixed Polarity
"           Fitter Options:     Keep Preassignments
"           Simulate Options:   Table Format, Register Powerup 0,
"                               X-Value 0, Z-Value 1,
"                               Brief Trace, Use .tmv File
"           Defaults:           Auto Update, Force Fit Update, Program Pause
"
" Anmerkung: Die gew hlte Pinbelegung erm glicht dem Fitter ein 100prozentiges
"           Ergebnis, welches bei automatischer Pinvergabe oder blockweiser
"           Zuordnung nicht mehr erreicht wird.
"           Des weiteren dienen die hier integrierten Testvektoren nur dem
"           funktionalen Nachvollziehen des Designs und erheben keinen Anspruch
"           auf eine 100%ige Testabdeckung.

PROF_REP device  'MACH110';
AMDMACH property 'COMMAND 1 _EG';

        " Eing nge "
        CLK_FIX, CLK_VAR                pin 35, 13;
        RXD_A, RXD_B                    pin 10, 11;

        " Ausg nge "
        CLK_VARO                        pin 24  istype 'reg, buffer';
        EN_A                            pin 40  istype 'com, buffer';
        EN_B                            pin 38  istype 'com, buffer';
        TXD                             pin 28  istype 'reg, invert';

        " Reset-Generator "
        IRES, IRES_O                    pin  5, 2  istype 'com, buffer';

        " Oszillator "
        OSZ_I, OSZ_O                    pin 33, 14 istype 'com, buffer';

        " digitale Filter "
        CHA_1, RXD_FA                   pin 36, 39 istype 'reg, buffer';
        CHB_1, RXD_FB                   pin 25, 27 istype 'reg, buffer';

        " Vorselektion "
        VS_ACTV, SEL_AB                 pin 15, 21 istype 'reg, buffer';

        " Bitregenerierung "
        REG_3                           pin 20 istype 'reg, buffer';
        REG_2                           pin 19 istype 'reg, buffer';
        REG_1                           pin  8 istype 'reg, buffer';
        REG_0                           pin  4 istype 'reg, buffer';
        DTR                             pin  3 istype 'com, buffer';

        " Timersteuerung "
        TCO_2, TCO_1, TCO_0             pin 18, 16, 6  istype 'reg, buffer';
        TAQ_2, TAQ_1                    pin 30, 37     istype 'reg, invert';
                      TAQ_0             pin         41 istype 'reg, buffer';
        TBQ_2,        TBQ_0             pin 29,     26 istype 'reg, invert';
               TBQ_1                    pin 31         istype 'reg, buffer';

declarations
        " Zustandsdeklarationen digitale Filter "
        FIL_A_STATES = [RXD_FA,CHA_1];
```

```
        FIL_B_STATES = [RXD_FB,CHB_1];
        FIL_A_VERGL  = [RXD_FA.fb,CHA_1.fb];
        FIL_B_VERGL  = [RXD_FB.fb,CHB_1.fb];
        FIL_Z00 = [0,0];
        FIL_Z01 = [0,1];
        FIL_Z10 = [1,0];
        FIL_Z11 = [1,1];
        " Vorsteuerung "
        VS_STATES = [VS_ACTV,SEL_AB];
        VS_VERGL = [VS_ACTV.fb,SEL_AB.fb];
        VS_NEUT = [0,0];
        VS_AACT = [1,1];
        VS_BACT = [1,0];
        VS_UDEF = [0,1];
        " Zustandsdeklarationen Bitregenerierung "
        REG_STATES = [REG_3,REG_2,REG_1,REG_0];
        REG_VERGL  = [REG_3.fb,REG_2.fb,REG_1.fb,REG_0.fb];
        REG_NEU = [0,0,0,0];
        REG_DLY = [0,0,0,1];
        REG_STA = [0,0,1,1];
        REG_BI0 = [0,0,1,0];
        REG_BI1 = [0,1,1,0];
        REG_BI2 = [0,1,1,1];
        REG_BI3 = [0,1,0,1];
        REG_BI4 = [0,1,0,0];
        REG_BI5 = [1,1,0,0];
        REG_BI6 = [1,1,0,1];
        REG_BI7 = [1,1,1,1];
        REG_PTY = [1,1,1,0];
        REG_STP = [1,0,1,0];
        REG_AGN = [1,0,1,1];
        REG_RES = [1,0,0,1];
        REG_UDF = [1,0,0,0];
        " Zustandsdeklarationen Timersteuerung "
        TCO_STATES = [TCO_2,TCO_1,TCO_0];
        TCO_VERGL  = [TCO_2.fb,TCO_1.fb,TCO_0.fb];
        TCO_Z00 = [0,0,0];
        TCO_Z01 = [0,0,1];
        TCO_Z02 = [0,1,1];
        TCO_Z03 = [0,1,0];
        TCO_Z04 = [1,1,1];
        TCO_Z05 = [1,1,0];
        TCO_Z06 = [1,0,1];
        TCO_Z07 = [1,0,0];
equations
        " Taktzuordnungen "
        [RXD_FA, CHA_1].clk         = CLK_FIX;
        [RXD_FB, CHB_1].clk         = CLK_FIX;
        [VS_ACTV,SEL_AB].clk        = CLK_FIX;
        [TCO_2..TCO_0].clk          = CLK_FIX;
        [TAQ_2..TAQ_0].clk          = CLK_FIX;
        [TBQ_2..TBQ_0].clk          = CLK_FIX;
        [REG_3..REG_0].clk          = CLK_VAR;
         TXD.clk                    = CLK_VAR;
         CLK_VARO.clk               = CLK_FIX;
```

```
" Reset Deklarationen "
[RXD_FA, CHA_1].ar          = 0;
[RXD_FB, CHB_1].ar          = 0;
[REG_3..REG_0].ar           = (REG_VERGL==REG_RES) & VS_ACTV.fb;
[VS_ACTV,SEL_AB].ar         = (REG_VERGL==REG_RES) & VS_ACTV.fb;
[TCO_2..TCO_0].ar           = (REG_VERGL==REG_RES) & VS_ACTV.fb;
[TAQ_2..TAQ_0].ar           = 0;
[TBQ_2..TBQ_0].ar           = 0;
 TXD.ar                     = 0;
 CLK_VARO.ar                = 0;
" Preset Deklarationen "
[RXD_FA, CHA_1].ap          = 0;
[RXD_FB, CHB_1].ap          = 0;
[REG_3..REG_0].ap           = 0;
[VS_ACTV,SEL_AB].ap         = 0;
[TCO_2..TCO_0].ap           = 0;
[TAQ_2..TAQ_0].ap           = 0;
[TBQ_2..TBQ_0].ap           = 0;
 TXD.ap                     = 0;
 CLK_VARO.ap                = 0;

" Reset-Erzeugung "
IRES_O    = IRES;        " Mitkopplung f r Schmidt-Trigger-Funktion
IRES_O.oe = IRES;        " Ausgangstreiber nur bei High aktivieren

" Quarz-Oszillator "
OSZ_O = !OSZ_I;          "Gegenkopplung f r Schwingbedingung "

" Ausgangssignale "
DTR    = (!(REG_VERGL==REG_NEU)
        & !(REG_VERGL==REG_DLY)
        & !(REG_VERGL==REG_RES));

EN_A   = (VS_VERGL==VS_BACT) & DTR;  "Vorsteuerung lauscht auf B
                                     "und Bitregenerierung ist
                                     "aktiv

EN_B   = (VS_VERGL==VS_AACT) & DTR;  "Vorsteuerung lauscht auf A
                                     "und Bitregenerierung ist
                                     "aktiv

!TXD :=    (((VS_VERGL==VS_AACT) & RXD_FA.fb)  "Daten entsprechend
           #((VS_VERGL==VS_BACT) & RXD_FB.fb));"der Vorsteuerung
                                               "abtasten und ausgeben
                  "TXD ist wegen der Invertierung von RXD_Fx negiert!

CLK_VARO := IRES&((TCO_VERGL==TCO_Z00)&!CLK_VARO.fb" hoher Takt, wenn inaktiv,
         # (TCO_VERGL==TCO_Z01)&TAQ_0.fb&TAQ_1.fb&TAQ_2.fb   " sonst nur bei
         # (TCO_VERGL==TCO_Z02)&TBQ_0.fb&TBQ_1.fb&TBQ_2.fb   "  berlauf
         # (TCO_VERGL==TCO_Z03)&TBQ_0.fb&TBQ_1.fb&TBQ_2.fb   " der Vorteiler
         # (TCO_VERGL==TCO_Z04)&TAQ_0.fb&TAQ_1.fb&TAQ_2.fb); " takten

" Vorteiler A und B "
"==================="
" Die Vorteiler A und B sind so ausgelegt, daa sie beim Start bereits
" mit etwa halbem Z hlwert vorgeladen sind und damit die Zeit bis zum
"  berlauf nur eine halbe Periode dauert. Der Vorladewert f r
" Vorteiler A betr gt beim ersten Mal n/2+2 also 110b, bei allen
" folgenden F llen 101b, da beim ersten Mal ein Takt f r die Bit-
" regenerierung und ein Takt f r die Timersteuerung verbraucht wurde,
```

```
    " w hrend bei laufender  bertragung nur ein Takt Verz gerung in der
    " Timersteuerung ben tigt wird. Durch diesen Mechanismus wird jedes
    " zu  bertragende Bit genau in der Mitte abgetastet.
    " Die Wahl der Polarit t der Ausgangssignale entspricht dem Wert der
    " Vorladewerte, da so keine weiteren Initialisierungsschritte n tig
    " sind.
   TAQ_0 := (IRES &  TCO_0.fb & !TAQ_0.fb                   "Togglebedingung
          # IRES & !TCO_0.fb & !(TCO_VERGL==TCO_Z00) & 1 "Preset ab
          # !IRES & 0);                                     "dem 2. Mal

  !TAQ_1 :=!(IRES &  TCO_0.fb &  TAQ_0.fb & !TAQ_1.fb      "Togglebedingung
           # IRES &  TCO_0.fb & !TAQ_0.fb &  TAQ_1.fb      "Haltebedingung
           # IRES & (TCO_VERGL==TCO_Z00)  & 1               "Preset nur beim
           #!IRES &  1);                                    "ersten Mal

  !TAQ_2 :=!(IRES& TCO_0.fb&  TAQ_0.fb&TAQ_1.fb &!TAQ_2.fb"Togglebedingung
          #  IRES& TCO_0.fb&!(TAQ_0.fb&TAQ_1.fb)& TAQ_2.fb"Haltebedingung
          #  IRES&!TCO_0.fb&1                               "Immer Preset
          # !IRES&1);

  !TBQ_0 :=!(IRES &  TCO_1.fb & !TBQ_0.fb                  "Togglebedingung
          #  IRES & !TCO_1.fb &  1                          "Preload 1
          # !IRES & 1);

   TBQ_1 := (IRES &  TCO_1.fb &  TBQ_0.fb & !TBQ_1.fb      "Togglebedingung
          #  IRES &  TCO_1.fb & !TBQ_0.fb &  TBQ_1.fb      "Haltebedingung
          #  IRES & !TCO_1.fb &  0                          "Preload 0
          # !IRES & 0);

  !TBQ_2 :=!(IRES& TCO_1.fb&  TBQ_0.fb&TBQ_1.fb &!TBQ_2.fb"Togglebedingung
          #  IRES& TCO_1.fb&!(TBQ_0.fb&TBQ_1.fb)& TBQ_2.fb"Haltebedingung
          #  IRES&!TCO_1.fb& 1                              "Preload 1
          # !IRES&1);

state_diagram FIL_A_STATES
    "================================================="
    "        digitales Filter Kanal A                 "
    "================================================="

    " Ausgangssignal des digitalen Filters ist das Zustandsbit
    " RXD_FA, welches aufgrund geeigneter Wahl der Zustandskodierung
    " dem gefiltertem und invertiertem Eingangssignal RXD_A entspricht.

   state FIL_Z00:
    if (IRES  & !RXD_A)  then FIL_Z01
                         else FIL_Z00
   state FIL_Z01:
    if !IRES             then FIL_Z00;
       else if !RXD_A    then FIL_Z11;
                         else FIL_Z00;
   state FIL_Z11:
    if !IRES             then FIL_Z00;
       else if !RXD_A    then FIL_Z11;
                         else FIL_Z10;
   state FIL_Z10:
    if !IRES             then FIL_Z00;
       else if !RXD_A    then FIL_Z11;
                         else FIL_Z00;
```

```
state_diagram FIL_B_STATES
        "=================================================="
        "      digitales Filter Kanal B                    "
        "=================================================="

        " Ausgangssignal des digitalen Filters ist das Zustandsbit
        " RXD_FB, welches aufgrund geeigneter Wahl der Zustandskodierung
        " dem gefiltertem und invertiertem Eingangssignal RXD_B entspricht.

        state FIL_Z00:
          if (IRES  & !RXD_B)  then FIL_Z01
                               else FIL_Z00
        state FIL_Z01:
          if !IRES             then FIL_Z00;
             else if !RXD_B    then FIL_Z11;
                               else FIL_Z00;

        state FIL_Z11:
          if !IRES             then FIL_Z00;
             else if !RXD_B    then FIL_Z11;
                               else FIL_Z10;

        state FIL_Z10:
          if !IRES             then FIL_Z00;
             else if !RXD_B    then FIL_Z11;
                               else FIL_Z00;

state_diagram VS_STATES
        "=================================================="
        "           Vorselektion                           "
        "=================================================="

        " Die Vorselektion besteht aus drei Zust nden, von denen die
        " Zust nde VS_AACT und VS_BACT nur  ber den asynchronen Reset
        " verlassen werden k nnen. Im  brigen reagiert die Vorsteuerung
        " bereits auf die  bergangsbedingungen der digitalen Filter,
        " wodurch die Aktivierung im gleichen Takt wie RXD_Fx stattfindet.

        state VS_NEUT:
          if !IRES                                      then VS_NEUT;
             else if ((FIL_A_VERGL==FIL_Z01)&!RXD_A#RXD_FA.fb) then VS_AACT;
             else if ((FIL_B_VERGL==FIL_Z01)&!RXD_B#RXD_FB.fb) then VS_BACT;
                                                        else VS_NEUT;
        state VS_AACT:
          if !IRES             then VS_NEUT;
                               else VS_AACT;

        state VS_BACT:
          if !IRES             then VS_NEUT;
                               else VS_BACT;

        state VS_UDEF:         goto VS_NEUT;

state_diagram REG_STATES
        "=================================================="
        "             Bitregenerierung                     "
        "=================================================="

        " Die Bitregenerierung wird mit dem Startbit auf A oder B
        " aktiviert bzw reaktiviert und durchl uft dann 12 bzw 11
        " Zust nde, die den Abtastzeitpunkten entsprechen.
```

```
state REG_NEU:
if ( RXD_FA.fb # RXD_FB.fb) then REG_DLY; "Startbit auf A oder B
                            else REG_NEU; " sonst weiter lauern

state REG_DLY:      goto REG_STA;
state REG_STA:      goto REG_BI0;
state REG_BI0:      goto REG_BI1;
state REG_BI1:      goto REG_BI2;
state REG_BI2:      goto REG_BI3;
state REG_BI3:      goto REG_BI4;
state REG_BI4:      goto REG_BI5;
state REG_BI5:      goto REG_BI6;
state REG_BI6:      goto REG_BI7;
state REG_BI7:      goto REG_PTY;
state REG_PTY:      goto REG_STP;

state REG_STP:
if ( RXD_FA.fb # RXD_FB.fb) then REG_STA; "Startbit auf A oder B
                            else REG_RES; " Sonst Vorst. loesen

state REG_AGN:      goto REG_NEU;

state REG_RES:      goto REG_NEU;

state REG_UDF:      goto REG_NEU;

state_diagram TCO_STATES
    "==================================================="
    "                Timersteuerung                     "
    "==================================================="
    " die Timersteuerung startet und stoppt die Vorteiler 0 und 1.
    " Dies erfolgt durch geschickte Wahl der Zustandscodierung, indem
    " einzelne Bits direkt als Start/Stop-Information ausgewertet werden.

    state TCO_Z00:
    " Sobald die Bitregenerierung gestartet hat, startet auch die
    " Timersteuerung und damit der Vorteiler 0
    if (!IRES)                    then TCO_Z00;
    else if (REG_VERGL==REG_DLY)  then TCO_Z01;
                                  else TCO_Z00;

    state TCO_Z01:
    " Vorteiler 0 l uft jetzt. Wenn w hrend der  bertragung des
    " Stopbits eine neue Startbedingung erkannt wird, startet die
    " Timersteuerung auch den Vorteiler 1
    if (!IRES)                              then TCO_Z00;
    else if ((REG_VERGL==REG_STP)
            &(RXD_FA.fb&(VS_VERGL==VS_AACT)
            #RXD_FB.fb&(VS_VERGL==VS_BACT)))
                                            then TCO_Z02;
                                            else TCO_Z01;

    state TCO_Z02:
    " Vorteiler 0 und 1 laufen jetzt. Sobald das Stopbit
    "  bertragen wurde, wird der Vorteiler 0 wieder gestoppt.
    if (!IRES)                    then TCO_Z00;
    else if !(REG_VERGL==REG_STP) then TCO_Z03;
                                  else TCO_Z02;

    state TCO_Z03:
    " Vorteiler 1 l uft jetzt. Wenn w hrend der  bertragung des
    " Stopbits eine neue Startbedingung erkannt wird, startet die
```

```
     " Timersteuerung wieder den Vorteiler 0.
     if (!IRES)                          then TCO_Z00;
     else if ((REG_VERGL==REG_STP)
             &(RXD_FA.fb&(VS_VERGL==VS_AACT)
              #RXD_FB.fb&(VS_VERGL==VS_BACT)))  then TCO_Z04;
                                         else TCO_Z03;

     state TCO_Z04:
     " Vorteiler 0 und 1 laufen jetzt. Sobald das Stopbit
     "  bertragen wurde, wird der Vorteiler 1 wieder gestoppt.
     if (!IRES)                     then TCO_Z00;
     else if !(REG_VERGL==REG_STP)  then TCO_Z01;
                                    else TCO_Z04;
     " nicht geutzte Zust nde f hren zum neutralen Zustand.
     state TCO_Z05:          goto TCO_Z00;
     state TCO_Z06:          goto TCO_Z00;
     state TCO_Z07:          goto TCO_Z00;

test_vectors   " Test Oszillator und Reset-Erzeugung "
               "====================================="

               " Es sind nur Oszillator- und Reset-Logik autark,
               " die  brige mua im Zusammenhang mit den
               " Zustandsmaschinen getestet werden.
               " Hierzu werden beide Eingangszust nde getestet
     ([IRES, OSZ_I] -> [IRES_O, OSZ_O])
      [ 0  , 0    ] -> [.Z.   ,  1   ];
      [ 1  , 1    ] -> [ 1    ,  0   ];

test_vectors   " Test digitale Filter Kanal A und B sowie Vorsteuerung "
               "======================================================"

               " Der Test der digitalen Filter und Vorsteuerung beginnt mit
               " dem R cksetzen der Logik mittels IRES-Signal. Anschliessend
               " werden der Reihe nach folgende Zust nde von Kanal A und B
               " durchlaufen:
               " FIL_Z00 -> FIL_Z00 -> FIL_Z01 -> FIL_Z00 -> FIL_Z01 ->
               "            FIL_Z11 -> FIL_Z11 -> FIL_Z10 -> FIL_Z11 ->
               "                 FIL_Z10 -> FIL_Z00
               " und nur auf Kanal B:
               " FIL_Z00 -> FIL_Z01 -> FIL_Z11 -> FIL_Z11 -> FIL_Z10 ->
               "            FIL_Z00 -> FIL_Z00
               "
               " Die Vorsteuerung mua beim  bergang von FIL_Z01 auf FIL_Z11
               " den angesprochenen Kanal durchschalten, wobei A Priorit t hat.
   ([CLK_FIX,IRES,RXD_A,RXD_B] -> [RXD_FA,CHA_1,RXD_FB,CHB_1,VS_ACTV,SEL_AB])
    [.C.    ,  0 , 1   , 1   ] -> [ 0    , 0   , 0    , 0   , 0     , 0    ];
    [.C.    ,  1 , 1   , 1   ] -> [ 0    , 0   , 0    , 0   , 0     , 0    ];
    [.C.    ,  1 , 0   , 0   ] -> [ 0    , 1   , 0    , 1   , 0     , 0    ];
    [.C.    ,  1 , 1   , 1   ] -> [ 0    , 0   , 0    , 0   , 0     , 0    ];
    [.C.    ,  1 , 0   , 0   ] -> [ 0    , 1   , 0    , 1   , 0     , 0    ];
    [.C.    ,  1 , 0   , 0   ] -> [ 1    , 1   , 1    , 1   , 1     , 1    ];
    [.C.    ,  1 , 0   , 0   ] -> [ 1    , 1   , 1    , 1   , 1     , 1    ];
    [.C.    ,  1 , 1   , 1   ] -> [ 1    , 0   , 1    , 0   , 1     , 1    ];
    [.C.    ,  1 , 0   , 0   ] -> [ 1    , 1   , 1    , 1   , 1     , 1    ];
    [.C.    ,  1 , 1   , 1   ] -> [ 1    , 0   , 1    , 0   , 1     , 1    ];
    [.C.    ,  1 , 1   , 1   ] -> [ 0    , 0   , 0    , 0   , 1     , 1    ];
   " und jetzt Kanal B aktivieren "
    [.C.    ,  0 , 1   , 1   ] -> [ 0    , 0   , 0    , 0   , 0     , 0    ];
```

```
    [.C.    ,  1 , 1   , 0   ] -> [ 0   , 0   , 0   , 1   , 0   , 0   ];
    [.C.    ,  1 , 1   , 0 ] ->[ 0    , 0   , 1    , 1   , 1   , 0  ];
    [.C.    ,  1 , 1   , 0   ] -> [ 0    , 0   , 1    , 1   , 1   , 0  ];
    [.C.    ,  1 , 1   , 1   ] -> [ 0    , 0   , 1    , 0   , 1   , 0   ];
    [.C.    ,  1 , 1   , 1 ] -> [ 0    , 0   , 0    , 0   , 1   , 0   ];
    [.C.    ,  0 , 1   , 1 ] ->[ 0    , 0   , 0    , 0   , 0   , 0   ];

test_vectors  " Test Bitregenerierung und Timersteuerung "
              "========================================="

              " Der Test erfolgt, indem zun chst die Logik  ber IRES
              " zur ckgesetzt und dann nacheinander folgende Zust nde
              " durchlaufen werden:
              " REG_NEU -> REG_DLY -> REG_STA -> REG_BI0 -> REG_BI1 ->
              "            REG_BI2 -> REG_BI3 -> REG_BI4 -> REG_BI5 ->
              "            REG_BI6 -> REG_BI7 -> REG_PTY -> REG_DLY -> REG_STA
              " Parallel dazu mua die Timersteuerung folgende Zust nde
              " annehmen:
              " TCO_Z00 -> TCO_Z01 -> TCO_Z02
([CLK_FIX,CLK_VAR,IRES,RXD_A,RXD_B]->
           [[REG_3..REG_0],[TCO_2..TCO_0],[TAQ_2..TAQ_0],[TBQ_2..TBQ_0],
                                            CLK_VARO,DTR,EN_A,EN_B,TXD])
 [.C.,.C.,0 , 1 , 1 ]->[ 0 , 0 , 6 , 5 , 0 , 0 , 0 , 0, 1 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 0 , 0 , 6 , 5 , 1 , 0 , 0 , 0,.X.];
 [.C.,.C.,1 , 0 , 0 ]->[ 0 , 0 , 6 , 5 , 0 , 0 , 0 , 0,.X.];
 " Jetzt hat die Vorsteuerung reagiert "
 [.C.,.C.,1 , 0 , 0 ]->[ 1 , 0 , 6 , 5 , 1 , 0 , 0 , 0,.X.];
 [.C.,.C.,1 , 0 , 0 ]->[ 3 , 1 , 6 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 2 , 1 , 7 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 6 , 1 , 0 , 5 , 1 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 7 , 1 , 1 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 5 , 1 , 2 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 4 , 1 , 3 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[12 , 1 , 4 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[13 , 1 , 5 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[15 , 1 , 6 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[14 , 1 , 7 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[10 , 1 , 0 , 5 , 1 , 1 , 0 , 1, 0 ];
 " jetzt kommt das 2. Startbit
 [.C.,.C.,1 , 0 , 0 ]->[ 3 , 3 , 1 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 2 , 2 , 2 , 6 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 6 , 2 , 5 , 7 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 7 , 2 , 5 , 0 , 1 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 5 , 2 , 5 , 1 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[ 4 , 2 , 5 , 2 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[12 , 2 , 5 , 3 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[13 , 2 , 5 , 4 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[15 , 2 , 5 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[14 , 2 , 5 , 6 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 0 ]->[10 , 2 , 5 , 7 , 0 , 1 , 0 , 1, 0 ];
 " jetzt kommt das 3. Startbit
 [.C.,.C.,1 , 0 , 1 ]->[ 3 , 7 , 5 , 0 , 1 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[ 2 , 1 , 6 , 1 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[ 6 , 1 , 7 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[ 7 , 1 , 0 , 5 , 1 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[ 5 , 1 , 1 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[ 4 , 1 , 2 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[12 , 1 , 3 , 5 , 0 , 1 , 0 , 1, 0 ];
 [.C.,.C.,1 , 0 , 1 ]->[13 , 1 , 4 , 5 , 0 , 1 , 0 , 1, 0 ];
```

```
[.C.,.C.,1 , 0 , 1 ]->[15 , 1 , 5 , 5 , 0 , 1 , 0 , 1, 0 ];
[.C.,.C.,1 , 0 , 1 ]->[14 , 1 , 6 , 5 , 0 , 1 , 0 , 1, 0 ];
[.C.,.C.,1 , 0 , 1 ]->[10 , 1 , 7 , 5 , 0 , 1 , 0 , 1, 0 ];
" jetzt kommt das 4. Startbit
[.C.,.C.,1 , 0 , 0 ]->[ 3 , 3 , 0 , 5 , 1 , 1 , 0 , 1, 0 ];
[.C.,.C.,1 , 1 , 1 ]->[ 2 , 2 , 1 , 6 , 0 , 1 , 0 , 1, 0 ];
[.C.,.C.,1 , 1 , 1 ]->[ 6 , 2 , 5 , 7 , 0 , 1 , 0 , 1, 0 ];
[.C.,.C.,1 , 1 , 1 ]->[ 7 , 2 , 5 , 0 , 1 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[ 5 , 2 , 5 , 1 , 0 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[ 4 , 2 , 5 , 2 , 0 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[12 , 2 , 5 , 3 , 0 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[13 , 2 , 5 , 4 , 0 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[15 , 2 , 5 , 5 , 0 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[14 , 2 , 5 , 6 , 0 , 1 , 0 , 1, 1 ];
[.C.,.C.,1 , 1 , 1 ]->[10 , 2 , 5 , 7 , 0 , 1 , 0 , 1, 1 ];
" jetzt kommt der asynchrone Reset
[.C.,.C.,1 , 1 , 1 ]->[ 0 , 0 , 5 , 0 , 1 , 0 , 0 , 0, .X.];
End
```

FITTER FILE

Note: This file has been condensed in order to save trees.

```
Flags Used:          Unplace=False          Max Packing=False
Flags Used:        Expand Small=True        Expand All=True

Reading Device Database ...
*****************************************************************************
     Mach PLD Fitter - v 1.46          Selbststeuernder Profibusrepeater
*****************************************************************************

*** Timing Analysis for Signals

 Parameter    Min    Max           Signal List (Those having Max delay.)
      Tpd      1      1              IRES_O           OSZ_O
      Tsu      1      1              CHA_1            RXD_FA           VS_ACTV
                                     SEL_AB           CHB_1            RXD_FB
                                   CLK_VARO           TBQ_1
      Tco      0      2               EN_A             EN_B
      Tcr      1      1                TXD           VS_ACTV           SEL_AB
                                     REG_3            REG_1            REG_0
                                     TCO_2         CLK_VARO
.

 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Register clock to combinatorial output
 Tcr - Register thru combinatorial logic to setup
 All delay values are expressed in terms of array passes

*** Device Resource Checks

                   Available        Used         Remaining
        Clocks:       2               2             0
          Pins:      38              32             6      ->      84%
     I/O Macro:      32              26             6
   Total Macro:      32              26             6
 Product Terms:     128              76             8      ->      92%

MACH-PLD Resource Checks OK!

Partitioning Design into Blocks...

*** Last Equations Placed in Blocks

Weakly -
Assign -

*** Block Partitioning Results

                 Array    Macros     # I/O    Buried   Product     Signal
                Inputs    Remain     Macro     Logic    Terms      Fanout
   Block-> A      17        4         12         0       64          15
   Block-> B      20        2         14         0       56          15

*** Block Signal List

Block-> A          TCO_0          TCO_1          TCO_2          REG_0
                   REG_1          REG_2          REG_3         SEL_AB
                 VS_ACTV            DTR          OSZ_O         IRES_O
```

```
Block-> B          TBQ_2           TBQ_1           TBQ_0           TAQ_2
                   TAQ_1           TAQ_0        CLK_VARO           CHB_1
                  RXD_FB           CHA_1          RXD_FA             TXD
                    EN_B            EN_A

|> INFORMATION F050 - Device Utilization....... *: 86 %

*** Feedback Map - Selbststeuernder Profibusrepeater

 Gbl Inp .--.      I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|     IRES : 0|     |21| OSZ_I          IRES : 0|     |21| /TBQ_0
         | 1|    CHB_1 : 1|     |20|               CHB_1 : 1|     |20| /TAQ_1
         | 2|    REG_0 : 2|     |19|                 DTR : 2|     |19|
         | 3|   RXD_FB : 3|     |18|              RXD_FB : 3|     |18| CLK_VAR
         | 4|    TCO_0 : 4|     |17| RXD_B         TCO_0 : 4|     |17| RXD_B
         | 5|          | 5|     |16| RXD_A        /TBQ_2 : 5|     |16| RXD_A
         '--'    REG_1 : 6|     |15: SEL_AB       /TAQ_2 : 6|     |15: SEL_AB
                       | 7|     |14: REG_3         TBQ_1 : 7|     |14|
                 CHA_1 : 8|     |13: REG_2         CHA_1 : 8|     |13: TAQ_0
               VS_ACTV : 9|     |12: TCO_2       VS_ACTV : 9|     |12: TCO_2
                 TCO_1 :10|     |11: RXD_FA        TCO_1 :10|     |11: RXD_FA
                       '--+-u--u+--'                     '--+-u--u+--'

*** Logic Map - Selbststeuernder Profibusrepeater

 Gbl Inp .--.      I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
    RXD_A| 0|   IRES_O | 0| 1   |21|             CLK_VARO  0| 4   |21|
    RXD_B| 1|      DTR | 1| 3   |20|               CHB_1 | 1| 1   |20|
  CLK_VAR| 2|    REG_0 | 2| 6   |19|              /TBQ_0 | 2| 1   |19|
         | 3|          | 3| *   |18|              RXD_FB | 3| 3   |18|
    OSZ_I| 4|    TCO_0 | 4| 6   |17|                /TXD | 4| 2   |17|
  CLK_FIX| 5|          | 5| *   |16|              /TBQ_2 | 5| 3   |16|
         '--'    REG_1 | 6| 6  3|15| SEL_AB       /TAQ_2 | 6| 3  .|15|
                       | 7| *  3|14| REG_3         TBQ_1 | 7| 2  .|14|
                 OSZ_O | 8| 1  3|13| REG_2         CHA_1 | 8| 1  3|13| TAQ_0
               VS_ACTV | 9| 5  3|12| TCO_2        /TAQ_1 | 9| 4  1|12| EN_A
                 TCO_1 |10| 4  *|11|                EN_B |10| 1  3|11| RXD_FA
                       '--+-u--u+--'                     '--+-u--u+--'
```

```
*** Pin Map - Selbststeuernder Profibusrepeater



                                 IRES_O
                                 DTR  |        .
                            REG_0  |  |        |  .
                          IRES  |  |  |        |  |  TAQ_0
                      TCO_0  |  |  |  |        |  |  |  EN_A
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
                    | 7                                   39|RXD_FA
               REG_1| 8                  G  V             38|EN_B
                    | 9                  n  c             37|/TAQ_1
               RXD_A|10                  d  c             36|CHA_1
               RXD_B|11                                   35|CLK_FIX
                Gnd |12            MACH-110               34| Gnd
             CLK_VAR|13                                   33|OSZ_I
               OSZ_O|14             V  G                  32|
             VS_ACTV|15             c  n                  31|TBQ_1
               TCO_1|16             c  d                  30|/TAQ_2
                    |17                                   29|/TBQ_2
                    |   1  1  2  2  2  2  2  2  2  2  2     |
                    |   8  9  0  1  2  3  4  5  6  7  8     |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                    TCO_2  |  |  |        |  |  |  |  /TXD
                       REG_2  |  |        |  |  |  RXD_FB
                          REG_3  |        |  |  /TBQ_0
                            SEL_AB        |  CHB_1
                                          CLK_VARO
The Design Doc is  stored in ===> prof_rep.Rpt
The Jedec Data is  stored in ===> prof_rep.Jed
The Placements are stored in ===> prof_rep.Plc
The  Fuse Plot is  stored in ===> prof_rep.Xpt

%% FITR %% Error Count: 0, Warning Count: 0
%% FITR %% File Processed Successfully. - File: prof_rep

$DEVICE MACH110 fit
$PINS 0

-------------------------------------------------------------
```

PLACEMENT FILE

```
; Flags Used:          Unplace=False         Max Packing=False
; Flags Used:       Expand Small=True         Expand All=True

; FITR generated placements

Pin    35        CLK_FIX  ; Inp         ; I  5
Pin    13        CLK_VAR  ; Inp         ; I  2
Pin    10          RXD_A  ; Inp         ; I  0
Pin    11          RXD_B  ; Inp         ; I  1
Node   18     CLK_VARO_I    Reg         ; B  0
Pin    40           EN_A   Comb         ; B 12
Pin    38           EN_B   Comb         ; B 10
Pin    28           /TXD    Reg         ; B  4
Pin     5           IRES  ; Inp         ; A  3
Pin     2         IRES_O   Comb         ; A  0
Pin    33          OSZ_I  ; Inp         ; I  4
Pin    14          OSZ_O   Comb         ; A  8
Node   26        CHA_1_I    Reg         ; B  8
Node   29       RXD_FA_I    Reg         ; B 11
Node   19        CHB_1_I    Reg         ; B  1
Node   21       RXD_FB_I    Reg         ; B  3
Node   11      VS_ACTV_I    Reg         ; A  9
Node   17       SEL_AB_I    Reg         ; A 15
Node   16        REG_3_I    Reg         ; A 14
Node   15        REG_2_I    Reg         ; A 13
Node    8        REG_1_I    Reg         ; A  6
Node    4        REG_0_I    Reg         ; A  2
Pin     3            DTR   Comb         ; A  1
Node   14        TCO_2_I    Reg         ; A 12
Node   12        TCO_1_I    Reg         ; A 10
Node    6        TCO_0_I    Reg         ; A  4
Node   24       /TAQ_2_I    Reg         ; B  6
Node   27       /TAQ_1_I    Reg         ; B  9
Node   31        TAQ_0_I    Reg         ; B 13
Node   23       /TBQ_2_I    Reg         ; B  5
Node   20       /TBQ_0_I    Reg         ; B  2
Node   25        TBQ_1_I    Reg         ; B  7
Pin    39         RXD_FA    Reg         ; B 11
Pin    36          CHA_1    Reg         ; B  8
Pin    27         RXD_FB    Reg         ; B  3
Pin    25          CHB_1    Reg         ; B  1
Pin    15        VS_ACTV    Reg         ; A  9
Pin    21         SEL_AB    Reg         ; A 15
Pin    20          REG_3    Reg         ; A 14
Pin    19          REG_2    Reg         ; A 13
Pin     8          REG_1    Reg         ; A  6
Pin     4          REG_0    Reg         ; A  2
Pin    18          TCO_2    Reg         ; A 12
Pin    16          TCO_1    Reg         ; A 10
Pin     6          TCO_0    Reg         ; A  4
Pin    24       CLK_VARO    Reg         ; B  0
Pin    41          TAQ_0    Reg         ; B 13
Pin    37         /TAQ_1    Reg         ; B  9
Pin    30         /TAQ_2    Reg         ; B  6
Pin    26         /TBQ_0    Reg         ; B  2
Pin    31          TBQ_1    Reg         ; B  7
Pin    29         /TBQ_2    Reg         ; B  5
```

```
; Group Mach_Seg_A TCO_0 TCO_1 TCO_2 REG_0 REG_1 REG_2 REG_3 SEL_AB VS_ACTV DTR OSZ_O
IRES_O
; Group Mach_Seg_B TBQ_2 TBQ_1 TBQ_0 TAQ_2 TAQ_1 TAQ_0 CLK_VARO CHB_1 RXD_FB CHA_1 RXD_FA
TXD EN_B EN_A
```

Replacement of MC68000 Processor with MC68020 or MC68030 with Coprocessor Option for MC68881 and MC68882: MACH110

Advanced Micro Devices

Application Note

by Christoph Niessen, Germany

INTRODUCTION

The MC68000 processor is one of the most widely used processors in minicomputers[1], peripherals[2] and controllers. The new processors in this family (MC68010, MC68020, MC68030 and MC68040) are of course, software compatible, but in addition it is not a trivial matter to integrate them into old hardware environments. After the 68020 for example, all buses have to be 32 bits wide and the processors can be expanded by a well thought-out coprocessor interface.

The new processors offer the following advantages:

- Higher clock frequency (up to 50 MHz for the MC68030)
- Doubled, external data buswidths (32 bit)
- Doubled, internal buswidths (32 bit)
- Fewer cycles per memory access (4 for the MC68000, 3 for the MC68020, 2 for the MC68030)[3]
- Improved microprogram with far fewer cycles per command
- Integrated cache

These advantages can be utilized only in part when a processor of this type is adapted to hardware that was designed only for the MC68000. The higher clock frequency and the doubled data buswidth for example, normally can not be implemented any more readily than the reduced number of clock cycles per memory access. The bus bandwidth of the MC68000 of 8 MHz amounts to a maximum of 16 bits*8 MHz/4 cycles = 4 Mbytes/s, in contrast to 32 bit*50 MHz/2 cycles = 100 Mbyte/s of an MC68030 at 50 MHz clock frequency. It is evident that the hardware of the computer can not keep pace with a processor of this type. For these reasons, the first attempts to replace the MC68000 were carried out only with the boundary conditions of equal clock frequency and 16-bit data buswidth. The first publications originate from the year 1985 from Motorola Corp. and for their adaptation, they needed the processors (MC68020/881) and an additional three PAL16R4A-15 and six resistors.

Other designs required up to 13 logic components. In spite of the expenses, the solutions could only be implemented at the thus yield enhanced performance of less than 50% in normal computation mode without FPU; in programs with FPU support, the performance rises considerably.

With the new, highly integrated logic components, like the MACH Chips by AMD, the entire circuit can be created in one component. With little extra expense, the following expansions can also be implemented:

- Bus interface for local expansions with the entire bus bandwidth
- Differing clock frequency for computer, microprocessor and coprocessor(s)
- Programmable (reduced) access times to parts of the computer that allow this

INCOMPATIBILITIES OF THE PROCESSORS

To replace an MC68000 by an MC68020, the differences in the buses have to be compensated. These differences consist of the following points:

- The MC68020 does not have a synchronous bus interface for the peripheral components of the MC68xxx family
- The MC68020 reads the data right in the third clock cycle of a data transfer
- A different protocol is used for read-modify-write cycles
- The autovector-interrupt protocol is different

MACH110

The MACH110 component by AMD is a programmable logic component with a typical PAL-structure. The complexity of two PAL22V16 is attained with two blocks of 16 macrocells each and three additional inputs. The 44-pin PLCC-housing takes up very little board space. Like all newer logic components, the macrocells are variously programmable (D- or T-flip-flop, inverter, OE...).

Now the pin-compatible MACH210 is twice as complex and can be used when the logic design will no longer fit into the MACH110-series after a "redesign."

[1] Atari, Commodore, Apple etc.

[2] Laser printer

[3] Minimal values

Publication# 17752 Rev. A Amendment/0
Issue Date: May 1993

ABEL

The ABEL development environment by DATA I/O offers the possibility to design, optimize and simulate

- Designs in equation form
- As state diagram or
- As value table

With an auxiliary module, these designs can then be "fitted" into a MACH component; that is, the allocation of logic to the component resources can be implemented.

DESIGN

The design consists essentially of a series of synchronous and asynchronous state machines. All state transitions are secured against hazards and races. The function of the single modules has been demonstrated with test vectors via the simulator. The processors MC68020 and MC68030 are compatible for the applications stated here, so that this design is suitable for both.

The MC68000 internally doubles the clock, so that the external signals will be activated with both clock flanks. For this reason, not only the normal processor clock, but also the inverted processor clock has to be fed to the MACH component. In order that the local bus is expandable, several lines have to be equipped with OK-drivers. For the processor and coprocessor, timing sources, perhaps also differing timing sources, will have to be made available.

Replacement of MC68000 Processor with MC68020 or MC68030 with Coprocessor Option for MC68881 and MC68882

Advanced Micro Devices

Application Note

by Christoph Neissen, Germany

EINLEITUNG

Der Prozessor MC68000 ist einer der am weitesten verbreiteten Prozessoren in Kleinrechnern[1], Peripheriegeraten[2] und Steuerungen. Die neuen Prozessoren dieser Familie (MC68010, MC68020, MC68030 und MC68040) sind zwar softwarekompatibel aber nicht trivial in alte Hardware–Umgebungen zu integrieren. Ab dem 68020 sind zum Beispiel alle Busse 32-Bit breit und die Prozessoren sind durch ein gut durchdachtes Koprozessorinterface erweiterbar.

Die neuen Prozessoren bieten folgende Vorteile:

- Höhere Taktfrequenz (bis zu 50 MHz beim MC68030)
- Doppelte externe Datenbusbreite (32-Bit)
- Doppelte interne Busbreiten (32-Bit)
- Weniger Zyklen pro Speicherzugriff (4 beim MC68000, 3 beim MC68020, 2 beim MC68030)[3]
- Verbessertes Mikroprogramm mit deutlich weniger Zyklen pro Befehl
- Integrierte Caches

Diese Vorteile lassen sich nur teilweise nutzen wenn ein solcher Prozessor an eine Hardware angepaßt wird, die nur für einen MC68000 konzipiert ist. Die höhere Taktfrequenz und die doppelte Datenbusbreite lassen sich zum Beispiel normalerweise ebensowenig implementieren wie die reduzierte Anzahl der Taktzyklen pro Speicherzugriff. Die Busbandbreite des MC68000 mit 8 MHz beträgt maximal 16 Bit*8 MHz/4 Zyklen=4 Mbyte/s im Gegensatz zu 32 Bit*50 MHz/2 Zyklen=100 Mbyte/s eines MC68030 bei 50 MHz Taktfrequenz. Es ist offensichtlich, daß die Hardware des Rechners mit einem solchen Prozessor nicht mithalten kann. Aus diesen Gründen wurden die ersten Versuche den MC68000 zu ersetzen nur mit den Randbedingungen gleiche Taktfrequenz und 16-Bit Datenbusbreite durchgeführt. Die erste Veröffentlichung stammt aus dem Jahr 1985 von MOTOROLA und benötigt für die Anpassung außer den Prozessoren (MC68020/881) noch 3 PAL16R4A–15 und 6 Widerstände. Andere Konzepte benötigen bis zu 13 Logikbausteine. Trotz des Aufwandes sind die Lösungen nur bei der Taktfrequenz des Rechners einsetzbar und bringen so Leistungssteigerungen von weniger als 50% im normalen Rechenbetrieb ohne FPU; bei Programmen mit FPU–Unterstützung steigt die Leistung dagegen erheblich.

Mit den neuen hochintegrierten Logikbausteinen wie den MACH-Chips von AMD ist die gesamte Schaltung in einem Baustein zu realisieren. Mit wenig Zusatzaufwand können auch noch folgende Erweiterungen implementiert werden:

- Busschnittstelle für lokale Erweitereungen mit der vollen Busbandbreite
- Unterschiedliche Taktfrequenz von Rechner, Mikroprozessor und Koprozessor(en)
- Programmierbare (reduzierte) Zugriffszeiten auf Teile des Rechners die dieses erlauben

INKOMPATIBILITÄTEN DER PROZESSOREN

Für den Ersatz eines MC68000 durch einen MC68020 müssen die Unterschiede der Busse ausgeglichen weden. Diese Unterschiede bestehen in folgenden Punkten:

- Der MC68020 hat kein synchrones Businterface für die Peripheriebausteine der MC68xx–Familie
- Der MC68020 liest bereits im dritten Taktzyklus eines Datentransfers die Daten
- Bei Read-modify-write-Zyklen wird ein anderes Protokoll benutzt
- Das Autovektor-Interrupt-Protokoll ist unterschiedlich

MACH110

Der Baustein MACH110 von AMD ist ein programmierbare Logikbaustein mit einer typischen PAL–Struktur. Mit zwei Blöcken mit je 16 Makrozellen und drei zusätzlichen Eingängen wird die Komplexität von 2 PAL22V16 erreicht. Das 44-pin PLCC-Gehäuse belegt nur wenig Platinenfläche. Wie bei allen neueren Logikbausteinen sind die Makrozellen vielfältig programmierbar (D-oder T-Flip-Flop, Inverter, OE...).

Mit dem Pin-kompatiblen MACH210 steht ein doppelt so komplexer Baustein zu Verfügung, der dann eingesetzt

[1] Atari, Commodore, Apple etc.

[2] Laserdrucker

[3] Minimalwerte

werden kann, wenn der Logikentwurf nach einem "Redesign" nicht mehr in den MACH110er paßt.

ABEL

Die Entwicklungsumgebung ABEL von DATA I/O bietet die Möglichkeit

- Designs in Gleichungsform,
- Als Zustandsdiagramm oder
- Als Wertetabelle

Zu enwerfen, optimieren und simulieren. Mit einem Zusatzmodul können diese Designs anschließend in einen MACH-Baustein "gefittet" werden, das bedeutet, daß die Zuordnung der Logik zu den Bausteinrecourcen durchgeführt wird.

ENTWURF

Der Entwurf besteht im Wesentlichen aus einer Reihe von synchronen und asynchronen Zustandmaschinen. Alle Zustandsübergänge sind gegen Hazards und Races gesichert. Die Funktion der einzelnen Module ist mit Testvektoren durch den Simulator nachgewiesen. Die Prozessoren MC68020 und MC68030 sind für die hier betroffenen Anwendungen kompatibel, so das dieser Entwurf für beide geeignet ist.

Der MC68000 verdoppelt intern den Takt, so daß die externen Signale mit beiden Taktflanken aktiviert werden. Aus diesem Grund muß dem MACH-Baustein nicht nur der normale Prozessortakt, sondern auch der invertierte zugeführt werden. Damit der lokale Bus erweiterbar ist, müssen einige Leitungen mit OK-Treibern versehen weden. Für den Prozessor und Koprozessor müssen—eventuell auch unterschiedliche—Taktquellen bereitgestellt werden.

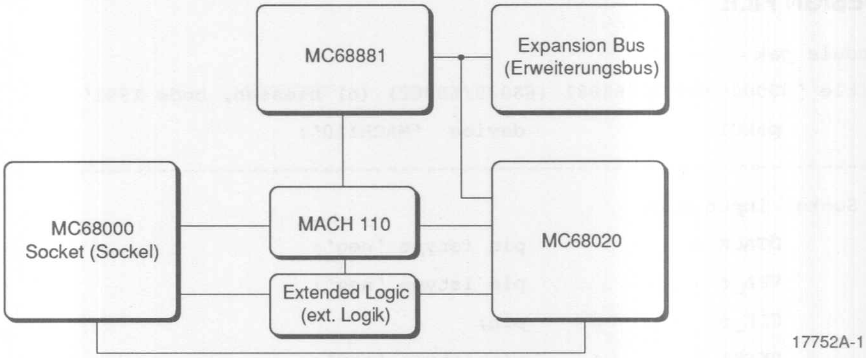


Figure 4-1. Block Diagram

DESIGN FILE

```
module pak
title '68000->68020/68881 (68030/68882) (c) niessen, bode 1991'
        pak01               device  'MACH110';
"----------------------------------------------------------------
" Socket input pins
        DTACK_s             pin istype 'neg';
        VPA_s               pin istype 'neg';
        CLK_s               pin;
        RESET_s             pin istype 'neg';
" Socket output pins
        AS_s                pin istype 'neg,reg';
        LDS_s               pin istype 'neg,reg';
        UDS_s               pin istype 'neg,reg';
        VMA_s               pin istype 'neg';
        BG_s                pin istype 'neg';
        E_p                 pin istype 'buffer,reg';
"----------------------------------------------------------------
" Processor input pins
        BG_p                pin istype 'neg';
        AS_p                pin istype 'neg';
        DS_p                pin istype 'neg';
        RnotW_p             pin;
        RMC_p               pin istype 'neg';
        SIZ1_p              pin;
        SIZ0_p              pin;
        FC2_p               pin;
        FC1_p               pin;
        FC0_p               pin;
        A31_p               pin;
        A24_p               pin;
        A19_p               pin;
        A18_p               pin;
        A17_p               pin;
        A16_p               pin;
        A15_p               pin;
        A14_p               pin;
        A13_p               pin;
```

```
        A0_p                pin;
" Processor output pins
        DSACK1_p            pin istype 'neg';
        AVEC_p              pin istype 'neg';
"---------------------------------------------------------------
        CLK_neg             pin;" CLK_s auf Platine invertiert
        CS_FPU              pin istype 'neg';" Output
        CS_RAM              node istype 'neg';
"---------------------------------------------------------------
        DSACK1_syn          node istype 'buffer,reg';
        DTERR               node;
        LDS                 node istype 'buffer,reg';
        UDS                 node istype 'buffer,reg';
        AS                  node istype 'buffer,reg';
"
" Zustandsregister fuer die Verzoegerung des Datenzugriffs:
"
        z1,z0               node istype 'buffer,reg';
        data_valid_00       =[z1,z0];
        Z0                  =[0,0];
        Z1                  =[0,1];
        Z2                  =[1,1];
        Z3                  =[1,0];
"
" Die folgenden Register werden fuer die E_p Erzeugung benutzt. Die
" Zustandskodierung ist so gewaehlt, dass alle Zustandsuebergaenge jeweils
" nur ein Bit aendern und dass das linke Bit direkt als E_p-Signal
" benutzt werden kann.
"
        c2,c1,c0            node istype 'buffer,reg';
        s0                  =[1,0,0,0];
        s1                  =[1,0,0,1];
        s2                  =[1,0,1,1];
        s3                  =[1,0,1,0];
        s4                  =[0,0,1,0];
        s5                  =[0,1,1,0];
        s6                  =[0,1,1,1];
        s7                  =[0,0,1,1];
        s8                  =[0,0,0,1];
```

```
        s9                  =[0,0,0,0];
        s10                 =[0,1,0,1];
        s11                 =[0,1,0,0];
        s12                 =[1,1,0,1];
        s13                 =[1,1,0,0];
        s14                 =[1,1,1,1];
        s15                 =[1,1,1,0];
"----------------------------------------------------------------
"AMDMACH property 'GROUP A LDS_s UDS_s AS_s BG_s';
AMDMACH property 'GROUP B DTERR DTACK_s z0 z1 SIZ0_p SIZ1_p A0_p';
AMDMACH property 'GROUP A VPA_s VMA_s E_p c2 c1 c0';
"----------------------------------------------------------------
        CPU_SPACE MACRO {(FC0_p&FC1_p&FC2_p)};
"       CP_COMMU  MACRO {([A19_p,A18_p,A17_p,A16_p]==[0,0,1,0])};
        CP_COMMU  MACRO {(!A31_p)};
        CP_ID     MACRO {([A15_p,A14_p,A13_p]==[0,0,1])};
        BGINH     MACRO {(CS_FPU#CS_RAM)};
"       IACKN     MACRO {(CPU_SPACE&AS_p&A19_p&A18_p&A17_p&A16_p)};
        IACKN     MACRO {(CPU_SPACE&AS_p&A31_p)};
        RESET_all MACRO {test_vectors(RESET_s->E_p);1->0;0->0;};
"----------------------------------------------------------------
equations
        CS_FPU=CPU_SPACE&CP_COMMU&CP_ID;
        CS_RAM=(!CPU_SPACE&!A31_p&A24_p);
        VMA_s.OE=!BGINH;
        BG_s=BG_p&!BGINH;BG_s.OE=BG_p&!BGINH;
        !AVEC_p=!IACKN#!VPA_s;
" VMA bestaetigt einen synchronen Zyklus der mit VPA eingeleitet wurde.
" VPA wird immer nur drei fallende Flanken vor E_p^ erkannt
        VMA_s=VPA_s&(([E_p.FB,c2.FB,c1.FB,c0.FB]==s11)#VMA_s);
        DSACK1_syn.clk=CLK_neg;
        DSACK1_syn.T=!AS_p&DSACK1_syn;" Ausschalten wenn AS_p inaktiv
"----------------------------------------------------------------
equations
" DTERR ist aktiv, wenn DTACK laenger als AS aktiv ist:
        DTERR=!AS_p&DTACK_s#DTERR&DTACK_s;
        AS:=AS_p&!DTERR#RMC_p;
        AS_s=AS&(AS_p#RMC_p);
        DSACK1_p=DSACK1_syn&AS_p;
```

```
        [UDS_s,LDS_s,AS_s].OE=!BGINH;
        [UDS,LDS,AS].CLK=CLK_s;
        LDS:=!DTERR&DS_p&(SIZ1_p#!SIZ0_p#A0_p);
        UDS:=!DTERR&DS_p&!A0_p;
        LDS_s=LDS&DS_p;
        UDS_s=UDS&DS_p;
"------------------------------------------------------------
equations
        data_valid_00.CLK=CLK_neg;
        data_valid_00.AR=RESET_s;
"------------------------------------------------------------
"
state_diagram data_valid_00
state Z0:if AS_s then Z1 else Z0;              "Start bei AS_s
state Z1:if DTACK_s then Z2 else Z1;           "DTACK abtasten
state Z2:goto Z3;                              "1 Zyklus verzoegern
state Z3:DSACK1_p=1;IF DS_p then Z3 else Z0;"Ende wenn DS inaktiv
"------------------------------------------------------------
equations
        [E_p,c2,c1,c0].CLK=CLK_neg;
        [E_p,c2,c1,c0].AR=RESET_s;
"------------------------------------------------------------
state_diagram [E_p,c2,c1,c0]
state s0:goto s1;
state s1:goto s2;
state s2:DSACK1_syn.T=VMA_s&AS_p&RnotW_p&!DSACK1_syn;goto s3; "Lesen
state s3:DSACK1_syn.T=VMA_s&AS_p&!RnotW_p&!DSACK1_syn;goto s4;"Schreiben
state s4:goto s5;
state s5:goto s6;
state s6:if VPA_s then s10 else s7;"synchron: erzeuge in 2 Zyklen VMA (s11)
state s7:goto s8;
state s8:goto s9;
state s9:goto s0;
"
state s10:goto s11;"VPA erkannt
state s11:goto s9;"VMA setzen
"
state s12:goto s0;
state s13:goto s0;
```

```
state s14:goto s0;
state s15:goto s0;
"-----------------------------------------------------------------
" Datenzugriff
RESET_all;
test_vectors
        ([CLK_s,CLK_neg,AS_p,DTACK_s,DS_p]->[data_valid_00,AS,AS_s,DSACK1_p]);
        [.x.,.x.,0,0,0]->[Z0,0,0,0];
        [.c.,.x.,0,0,0]->[Z0,0,0,0];
        [.x.,.x.,1,0,1]->[Z0,0,0,0];
        [.x.,.c.,1,0,1]->[Z0,0,0,0];
        [.x.,.x.,1,0,1]->[Z0,0,0,0];
        [.c.,.x.,1,0,1]->[Z0,1,1,0];" AS_p ist jetzt mit CLK synchronisiert
        [.x.,.x.,1,0,1]->[Z0,1,1,0];
        [.x.,.c.,1,0,1]->[Z1,1,1,0];" AS_s erkannt
        [.x.,.x.,1,0,1]->[Z1,1,1,0];
        [.c.,.x.,1,1,1]->[Z1,1,1,0];
        [.x.,.x.,1,1,1]->[Z1,1,1,0];
        [.x.,.c.,1,1,1]->[Z2,1,1,0];" DTACK_s erkannt aber erst im
        [.x.,.x.,1,1,1]->[Z2,1,1,0];" naechsten Zyklus weitergeben
        [.c.,.x.,1,1,1]->[Z2,1,1,0];
        [.x.,.x.,1,1,1]->[Z2,1,1,0];
        [.x.,.c.,1,1,1]->[Z3,1,1,1];" DSACK1 aktiviert
        [.c.,.x.,1,1,1]->[Z3,1,1,1];
        [.x.,.c.,1,1,1]->[Z3,1,1,1];
        [.x.,.x.,0,1,0]->[Z3,1,0,1];
        [.c.,.x.,0,1,0]->[Z3,0,0,1];
        [.x.,.c.,0,0,0]->[Z0,0,0,0];" Ende weil DS_p inaktiv
        [.c.,.x.,0,0,0]->[Z0,0,0,0];
        [.x.,.c.,0,0,0]->[Z0,0,0,0];
        [.x.,.x.,0,0,0]->[Z0,0,0,0];
"-----------------------------------------------------------------
" DTERR-Test
RESET_all;
test_vectors([AS_p,DTACK_s]->DTERR);
        [0,0]->0;" Start
        [1,0]->0;" Adressen gueltig
        [1,1]->0;" Daten uebernommen
        [0,1]->1;" Adressen nicht mehr gueltig
```

```
        [0,1]->1;" Fehlerbedingung
        [0,0]->0;" ab jetzt wieder alles ok
        [1,0]->0;" neuer Zyklus
        [1,1]->0;
        [0,1]->1;
        [0,1]->1;
        [1,1]->1;" neuer AS muss maskiert werden
        [1,0]->0;" neuer AS ist erlaubt wenn DTACK fehlt
        [1,0]->0;
        [0,0]->0;" z. B. Busfehler
"--------------------------------------------------------------
RESET_all;
test_vectors([DS_p,SIZ1_p,SIZ0_p,A0_p] -> [LDS.D,UDS.D])
        [0,.x.,.x.,.x.] -> [0,0];
        [1,0,0,0]       -> [1,1];
        [1,0,0,1]       -> [1,0];
        [1,0,1,0]       -> [0,1];
        [1,0,1,1]       -> [1,0];
        [1,1,0,0]       -> [1,1];
        [1,1,0,1]       -> [1,0];
        [1,1,1,0]       -> [1,1];
        [1,1,1,1]       -> [1,0];
"--------------------------------------------------------------
" VMA-Test: best case read
RESET_all;
test_vectors"best case"([CLK_neg,AS_p,RnotW_p,!VPA_s]
                    -> [E_p,!VMA_s,DSACK1_syn])
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[0,1,0];
        [.c.,1,1,1]->[0,1,0];
        [.c.,1,1,1]->[0,1,0];
        [ 0 ,1,1,0]->[0,1,0];
        [.c.,1,1,0]->[0,1,0];" hier wird VPA_s getestet und erkannt...
        [.c.,1,1,0]->[0,0,0];" und in diesem Takt dann VMA_s aktiviert
        [.c.,1,1,0]->[0,0,0];
        [.c.,1,1,0]->[1,0,0];
```

```
        [.c.,1,1,0]->[1,0,0];
        [.c.,1,1,0]->[1,0,0];
        [.c.,1,1,0]->[1,0,1];" Datenuebernahme
        [.c.,1,1,0]->[0,0,1];
        [.x.,0,1,0]->[0,0,1];
        [.x.,0,0,1]->[0,1,1];
        [.x.,0,0,1]->[0,1,1];
        [.c.,0,0,1]->[0,1,0];
        [.c.,0,0,1]->[0,1,0];
"---------------------------------------------------------------
" VMA-Test: best case write
RESET_all;
test_vectors"best case"([CLK_neg,AS_p,!RnotW_p,!VPA_s]
                    -> [E_p,!VMA_s,DSACK1_syn])
        [.c.,0,0,1]->[1,1,0];
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[1,1,0];
        [.c.,1,1,1]->[0,1,0];
        [.c.,1,1,1]->[0,1,0];
        [.c.,1,1,1]->[0,1,0];
        [.c.,1,1,0]->[0,1,0];" hier wird VPA_s getestet
        [.c.,1,1,0]->[0,0,0];
        [.c.,1,1,0]->[0,0,0];
        [.c.,1,1,0]->[1,0,0];
        [.c.,1,1,0]->[1,0,0];
        [.c.,1,1,0]->[1,0,0];
        [.c.,1,1,0]->[1,0,0];
        [.c.,1,1,0]->[0,0,1]; " Datenuebernahme
        [.c.,0,0,1]->[0,1,0];
        [.c.,0,0,1]->[0,1,0];
"---------------------------------------------------------------
" VMA-Test: worst case
RESET_all;
test_vectors"worst case"([CLK_neg,!VPA_s] -> [E_p,!VMA_s])
        [.c.,1]->[1,1];
        [.c.,1]->[1,1];
        [.c.,1]->[1,1];
        [.c.,1]->[1,1];
```

```
        [.c.,1]->[0,1];
        [.c.,1]->[0,1];
        [.c.,1]->[0,1];
        [.c.,1]->[0,1];" hier wird VPA_s getestet
        [ 0 ,0]->[0,1];
        [.c.,0]->[0,1];
        [.c.,0]->[0,1];
        [.c.,0]->[1,1];
        [.c.,0]->[1,1];
        [.c.,0]->[1,1];
        [.c.,0]->[1,1];
        [.c.,0]->[0,1];
        [.c.,0]->[0,1];
        [.c.,0]->[0,1];
        [.c.,0]->[0,1];" hier wird nocheinmal getestet und diesmal erkannt
        [.c.,0]->[0,0];" also kann jetzt auch VMA_s aktiviert werden.
        [.c.,0]->[0,0];
        [.c.,0]->[1,0];
        [.c.,0]->[1,0];
        [.c.,0]->[1,0];
        [.c.,0]->[1,0];" Datenuebernahme
        [.c.,0]->[0,0];
        [.c.,1]->[0,1];
        [.c.,1]->[0,1];
"--------------------------------------------------------------
end
```

FITTER FILE

Note: This file has been condensed in order to save trees.

```
AMD MACH FITR - MARKET RELEASE (1-24-91)

(C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990

Flags Used:              Unplace=False         Max Packing=True

Flags Used:         Expand Small=False          Expand All=False

************************************************************************************

Mach PLD Fitter - v 1.4   68000->68020/68881 (68030/68882) (c) niessen, bode 1991

************************************************************************************

*** Timing Analysis for Signals

 Parameter   Min    Max          Signal List (Those having Max delay.)

     Tpd      1      2               AS_s            LDS_s            UDS_s

                                    VMA_s             BG_s

     Tsu      1      3                 z0        DSACK1_syn

     Tco      0      1           DSACK1_p            LDS_s            UDS_s

                                     AS_s            VMA_s

     Tcr      1      2                 z0        DSACK1_syn

 Key:

 Tpd - Combinatorial propagation delay, input to output

 Tsu - Combinatorial setup delay before clock

 Tco - Register clock to combinatorial output

 Tcr - Register thru combinatorial logic to setup

 All delay values are expressed in terms of array passes

*** Device Resource Checks

                Available        Used        Remaining

       Clocks:      2              2             0

         Pins:     38             30             8      ->     78%

    I/O Macro:     32              9            23

  Total Macro:     32             20            12

 Product Terms:   128             43            48      ->     62%

MACH-PLD Resource Checks OK!

Partitioning Design into Blocks...

*** Last Equations Placed in Blocks

Weakly -

Assign -            AS_s           UDS_s            BG_s         DSACK1_p

Assign -

*** Block Partitioning Results

               Array    Macros     # I/O    Buried   Product    Signal
```

```
                Inputs     Remain     Macro     Logic     Terms     Fanout
   Block-> A     22          6          5         5         40        11
   Block-> B     18          6          4         6         40         7
*** Block Signal List
Block-> A            LDS_s          CS_RAM          CS_FPU           AVEC_p
              DSACK1_syn             E_p             c0                  c1
                      c2           VMA_s
Block-> B         DSACK1_p            BG_s          UDS_s              AS_s
                      AS             UDS             LDS                 z0
                      z1           DTERR
|> INFORMATION F050 - Device Utilization....... *: 76 %
*** Feedback Map - 68000->68020/68881 (68030/68882) (c) niessen, bode 1991
 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|    FC2_p : 0|       |21| FC0_p      DTACK_s : 0|       |21| A0_p
         | 1|    FC1_p : 1|       |20| RESET_s            | 1|       |20| RESET_s
         | 2|      E_p : 2|       |19| A13_p        DTERR : 2|       |19|
         | 3|       c0 : 3|       |18| VMA_s           z0 : 3|       |18| /AS_s
         | 4|       c1 : 4|       |17| DS_p            z1 : 4|       |17| DS_p
         | 5|       c2 : 5|       |16| AS_p           UDS : 5|       |16| AS_p
         '--'  DSACK1_s  6|       |15: VPA_s     DSACK1_s  6|       |15: AS
                 CS_RAM : 7|       |14: LDS        CS_RAM : 7|       |14: RMC_p
                  A15_p : 8|       |13: RnotW_p            | 8|       |13: BG_p
                  A24_p : 9|       |12: A31_p      SIZ0_p : 9|       |12: SIZ1_p
                  A14_p :10|       |11: CS_FPU             |10|       |11: CS_FPU
                          '--+-u--u+--'                    '--+-u--u+--'
*** Logic Map - 68000->68020/68881 (68030/68882) (c) niessen, bode 1991
 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
     AS_p| 0|    LDS_s | 0| 1   |21|              UDS_s | 0| 1   |21|
     DS_p| 1|    VMA_s | 1| 2   |20|              /AS_s | 1| 2   |20|
  CLK_neg| 2|      E_p | 2| 3   |19|              DTERR | 2| 2   |19|
  RESET_s| 3|       c0 | 3| 3   |18|                 z0 | 3| 2   |18|
    FC0_p| 4|       c1 | 4| 4   |17|                 z1 | 4| 3   |17|
    CLK_s| 5|       c2 | 5| 3   |16|                UDS   5| 1   |16|
         '--'  DSACK1_s  6| 3  .|15|                   | 6|.  2|15| AS
                 CS_RAM | 7| 3  .|14|                  | 7|.  3|14| LDS
                 CS_FPU | 8| 1  .|13|             BG_s  8| 1  .|13|
                 AVEC_p | 9| 1  .|12|         DSACK1_p  9| 2  .|12|
                        |10| .  .|11|                  |10| .  .|11|
                        '--+-u--u+--'                  '--+-u--u+--'
```

```
*** Pin Map - 68000->68020/68881 (68030/68882) (c) niessen, bode 1991


                                  LDS_s
                               VMA_s  |        A15_p
                              E_p  |  |        |  A24_p
                       DTACK_s  |  |  |        |  |  A14_p
                           .  |  |  |  |       |  |  |  .
                           |  |  |  |  |       |  |  |  |
                    .-----'--'--'--'--'--o--'--'--'--'---.
                    |                        4  4  4  4  4|
                    |      6  5  4  3  2  1  4  3  2  1  0|
                    | 7                                 39|A31_p
                    | 8                     G  V        38|RnotW_p
                    | 9                     n  c        37|DSACK1_p
                AS_p|10                     d  c        36|BG_s
                DS_p|11                                 35|CLK_s
                 Gnd|12            MACH-110             34| Gnd
             CLK_neg|13                                 33|FC0_p
              CS_FPU|14                V  G             32|RESET_s
              AVEC_p|15                c  n             31|FC2_p
              SIZ0_p|16                c  d             30|FC1_p
               A13_p|17                                 29|
                    |   1  1  2  2  2  2  2  2  2  2  2   |
                    |   8  9  0  1  2  3  4  5  6  7  8   |
                    '---.--.--.--------------.--.--.--.---'
                        |  |  |  |           |  |  |  |  
                    VPA_s  |  |_|           |  |  ||  '
                       RMC_p  |_|           |  |  |  A0_p
                           BG_p  |           |  '
                            SIZ1_p           |  /AS_s
                                          UDS_s
```

The Design Doc is stored in ===> pak01.Rpt

The Jedec Data is stored in ===> pak01.Jed

The Placements are stored in ===> pak01.Plc

The Fuse Plot is stored in ===> pak01.Xpt

%% FITR %% Error Count: 0, Warning Count: 2

%% FITR %% File Processed Successfully. - File: pak01

PLACEMENT FILE

```
; Flags Used:               Unplace=False          Max Packing=True
; Flags Used:          Expand Small=False           Expand All=False
; FITR generated placements
Pin     5          DTACK_s  ; Inp          ; A  3
Pin    18            VPA_s  ; Inp          ; A 12
Pin    35            CLK_s  ; Inp          ; I  5
Pin    32          RESET_s  ; Inp          ; I  3
Pin    25            /AS_s    Comb         ; B  1
Pin     2            LDS_s    Comb         ; A  0
Pin    24            UDS_s    Comb         ; B  0
Pin     3            VMA_s    Comb         ; A  1
Pin    36             BG_s    Comb         ; B  8
Node    4            E_p_I     Reg         ; A  2
Pin    20             BG_p  ; Inp          ; A 14
Pin    10             AS_p  ; Inp          ; I  0
Pin    11             DS_p  ; Inp          ; I  1
Pin    38          RnotW_p  ; Inp          ; B 10
Pin    19            RMC_p  ; Inp          ; A 13
Pin    21           SIZ1_p  ; Inp          ; A 15
Pin    16           SIZ0_p  ; Inp          ; A 10
Pin    31            FC2_p  ; Inp          ; B  7
Pin    30            FC1_p  ; Inp          ; B  6
Pin    33            FC0_p  ; Inp          ; I  4
Pin    39            A31_p  ; Inp          ; B 11
Pin    42            A24_p  ; Inp          ; B 14
Pin    43            A15_p  ; Inp          ; B 15
Pin    41            A14_p  ; Inp          ; B 13
Pin    17            A13_p  ; Inp          ; A 11
Pin    27             A0_p  ; Inp          ; B  3
Pin    37         DSACK1_p    Comb         ; B  9
Pin    15           AVEC_p    Comb         ; A  9
Pin    13          CLK_neg  ; Inp          ; I  2
Pin    14           CS_FPU    Comb         ; A  8
Node    9           CS_RAM    Comb         ; A  7
Node    8       DSACK1_syn     Reg         ; A  6
Node   20            DTERR    Comb         ; B  2
Node   32              LDS     Reg         ; B 14
```

```
Node  23            UDS    Reg         ; B  5
Node  33             AS    Reg         ; B 15
Node  22             z1    Reg         ; B  4
Node  21             z0    Reg         ; B  3
Node   7             c2    Reg         ; A  5
Node   6             c1    Reg         ; A  4
Node   5             c0    Reg         ; A  3
Pin    4            E_p    Reg         ; A  2
; Group Mach_Seg_A LDS_s CS_RAM CS_FPU AVEC_p DSACK1_syn E_p c0 c1 c2 VMA_s
; Group Mach_Seg_B DSACK1_p BG_s UDS_s AS_s AS UDS LDS z0 z1 DTERR
```

State Machine for a Transmitter/Receiver Module for High-Speed Serial Data Transmission: MACH110 and MACH210

Advanced Micro Devices

Application Note

by Günter Delfs, Deutsches Elektronen Synchrotron, Germany

GENERAL INFORMATION ON THE HSSL MODULE

HSSL –> High Speed Serial Link

In order to transfer high data rates over a fast, serial interface, a module was developed with 'AMD TAXI chips' that allows the optional transmission of bytes or 16-bit words. A transfer of 32-bit words is also readily possible through the expansion of the control function and the logic. This HSSL module (High Speed Serial Link) can be used in links between VME computer (16- or 32-bit words) and IBM(370) channels (bytes).

The serial output or input signals of the TAXI chips will be moved via lightwave senders or receivers and thus makes possible a tested lightwave conductor link of more than 3.5 km distance. The data transfer rate will be determined by the timing frequency of the TAXI chips and by the system management times.

Brief Functional Description of the HSSL Module

In the following brief description of the HSSL module, the concept of "control logic" of the sender/receiver will be used. This will include the MACH110/210, the PAL20RA10 (Figure 5-1) and the PALCE16V8 (Figure 5-2).

In the MACH110/210 the 'read/write-byte/word transfer' will be controlled with 'State Machine A' and the 'Requests' with 'State Machine B'.

The PAL20RA10 and the PALCE16V8 control the single signals of the TAXI chips (DSTR, CSTR, RCO..2, VLIN etc.), Go/Stop Requests, Errors etc. The design for the PAL20RA10 and PALCE16V8 will not be explained here.

The HSSL module is only the sender/receiver unit of a larger unit (call the "system" in the description). The STATES inserted into the description are intended to simplify reading and understanding of the MACH design.

For the MACH110, note that since nearly all input signals are asynchronous, they must be synchronized for a STATE MACHINE via a precircuited register. In the MACH210 the 'BURIED REGISTER' will be used for synchronization.

Both designs (MACH110/MACH210) are provided!

PREPARATION OF THE DESIGN FOR THE MACH110/210

Several input signals should be active 'low'. This is to be stated by 'invert' for the corresponding output signal in the 'declarations' column, e.g. with

.... pin istype 'reg_d,invert';

However, the 'fitter' then generates more product terms for this signal than are allowed in the MACH110/210. I have identified these signals with a ' _ ' and denoted them in the design in the declaration with ...istype 'neg'.

BRIEF DESCRIPTION OF THE HSSL MODULE

Initializing the HSSL Module

The sender FIFO (XMT-FIFO) will be deleted from the system with a reset signal (XRXMT - Reset XMT_FIFO) before every data transfer. The receiver FIFO (RCV-FIFO) will be deleted before every data transfer by a Reset-RCV signal (RCVRS) sent from the sender.

To do this, the sender moves the XRREM signal (Reset Remote) delivered from its system, to the control logic (PAL20RA10) that generates a 'send init' request (RQRR). The RQRR has the highest priority with respect to all other requests.

ñ state B02 ñ

(Refer to the design file for all state descriptions)

From the control logic, the command-bits XCI0 and XCI1 with the attendant STRB (strobe) signal will be sent via the MXT-TAXI chip to the receiver.

ñ state B10 ñ

ñ state B11 ñ

For the receiver, these two command bits will be recognized as RCI0 and RCI1 (PALCE16V8) and subsequently a reset-RCV-FIFO (RCVRS) will be sent from the PAL20RA10.

Data Transfer

Data can be transferred only when the channel to the sender and receiver has been released by the systems and the signal CHSL (channel select) is applied to the

Publication# 17754 Rev. A Amendment/0
Issue Date: May 1993

control logic. The transfer direction is specified with the signal CHRD (channel read) upon release of the channel.

Sender XMT ñ state A01 ñ

Receiver RCV ñ state A01 ñ

With the signal SW16 the data can be transferred optionally as byte (XD0...7) or as 16-bit word (XD 0...15). The signal SW16 can be hard wired (switch) or can be set with the release of the channel.

Send Data

Once the channel has been released and data is to be sent, the control logic will signal the system with the signal XRDY (data input free for write) that the data transfer can begin.

XMT ñ state A10 ñ

The sender data will be moved by the system with the signal WDAT (write data) into the sender latch (XMT latch) and at the same time, with the WDAT signal, the 'write cycle' will be started in the control logic.

The outputs of the XMT latch will be released byte by byte with the signals WEH (write enable high) or WEL (write enable low).

XMT ñ state A11 ñ

XMT ñ state A13 ñ

With the attendant signal WFI (write into XMT-FIFO), the data will be written into XMT-FIFO, provided the FIFO signals by its full-flag (FFR - Fifo free), that it is not yet "full."

XMT ñ state A12 ñ

XMT ñ state A14 ñ

The empty flag of the XMT-FIFOS sends a 'send data' request (RQDT) to the control logic right after the first WFI,

SMT ñ state B02 ñ

that then reads out the data from the FIFO with the XRD signal (read XMT-FIFO) and sends it with the attendant strobe signal STRB to the XMT-TAXI chip.

XMT ñ state B50 ñ

XMT ñ state B51 ñ

With the signal ACK, the XMT-TAXI chip confirms the data transfer. Additional data will be passed along to the XMT-TAXI chip only when ACK is removed again.

XMT ñ state B02 ñ

If the last byte or 16-bit word is transferred from the system, the signal XEOR (end of write data) with the data bits rests at the input of the XMT-FIFOS and will be transferred with the data.

The signal XEOR is the signal XXEOR generated by the system. It will be held until turn off of the channel and thus of the signal CHSL.

If the data transfer is completed, the sender receives a 'Transfer End' feedback from the receiver via the status bits. The system of the sender then shuts off the channel and the signal CHSL disappears. The control logic thus goes back into the waiting loop ('idle loop') and waits for the next CHSL.

XMT ñ state A01 ñ

Receive Data

The Receive Data will be moved from the output of the RCV-TAXI chips with the signal WDST (write received data into FIFO) into the Receive-FIFO (RCV-FIFO). The Empty Flag of the RCV-FIFOS then signals the control logic (sign signal FDA–fifo data available) that data is available.

RCV ñ state A20 ñ

If no data is being read out from the system from the receiver latch (RCV latch) and the signal RDT (RCV-FIFO data read out done) is thus not applied, the control logic will start the 'read cycle'.

RCV ñ state A20 ñ

The inputs from the RCV latch will be opened with the signals REH (read enable high) or REL (read enable low) byte by byte and at the same time, with the signal RFI (read data from RCV-FIFO) the data will be taken from the RCV-FIFO into the RCV latch.

RCV ñ state A21 ñ

RCV ñ state A22 ñ

RCV ñ state A23 ñ

If the data is in the RCV latch, then the control logic signals the system with the signal XRDY (read data available), that the data can be picked up.

RCV ñ state A24 ñ

The control logic now waits until the system confirms the data transfer with the signal RDT (RCV- FIFO data read out done). Only then will additional data be read out from the RCV-FIFO.

RCV ñ state A24 ñ

RCV ñ state A20 ñ

If the signal REOR (read end of record) is also read out from the RCV-FIFO with the data, then the control logic

will send the signal XEOR (end of read data) to the system.

The control logic now waits until the signal CHSL is removed and then goes back into the waiting loop ('idle loop').

RCV ñ state A30 ñ

RCV ñ state A01 ñ

Status

The channel need not be opened for the transfer of the status byte. An 8-bit status will be sent or received.

The transfer of the status has the second-highest priority. Sending of the status will be triggered by a change of one or more status bits (XXS 0...7). An 8-bit comparator ascertains the change and sends a 'send status' request (RQSTA) to the control logic.

ñ state B02 ñ

If no request of higher priority is pending (e.g. RQRR), then the RQSTA will be executed.

ñ state B20 ñ

The signal LST (latch xmitter status) transfers the changed status byte to the XMT-STATUS Register and to the comparator, that then takes the RQSTA away again.

With the transfer of the status byte to the XMT-TAXI chip by the STRB signal, an additional command bit (XCI2) will be sent, so that on the receiver side, the status bits are not taken into the RCV-FIFO, but rather in the receiving status register RCV-STATUS (XRS 0...7).

ñ state B21 ñ

Data Flow Control

The RCV-FIFO is relatively small (1K * 9 bit).

If data is being written into the RCV-FIFO faster than it is being read, then when the FCV-FIFO is half full, the half-full flag RHF (RHF -> RCV Half Full Flag) will be set. This then will signal the control logic that no more data can be received.

A 'send xoff' request RQSTP (STOP Request) will be subsequently generated that causes a command-bit transfer to the sender.

RCV ñ state B02 ñ

With the command bit XCI0 and the strobe signal STRB, this STOP command will be sent via the XMT-TAXI chip of the data receiver to the data sender.

RCV ñ state B30 ñ

RCV ñ state B31 ñ

By the received command-bit RCI0, the sender recognizes that no more data is to be sent. The control logic of the sender (PALCE16V8) then produces a STOPX signal that prevents an additional data transfer.

XMT ñ state B02 ñ

Now if the data receiver has read out its RCV-FIFO and if its half-full flag has been reset, then a 'send xon' request RQGO (GO Request) will be sent to the control logic.

RCV ñ state B02 ñ

With the command bit XCI1 and the strobe signal STRB, this GO command will be sent via the XMT-TAXI chip of the data receiver to the data sender.

RCV ñ state B40 ñ

RCV ñ state B41 ñ

The sender recognizes, based on the received command bit RCI1, that data can be sent again.

The control logic of the sender then picks up the STOPX signal again. The request RQDT will be released and data can be transferred.

XMT ñ state B02 ñ

XMT ñ state B50 ñ

XMT ñ state B51 ñ

Communication Between Sender and Receiver

At this point, the meanings of only the important status bits will be explained.

XMT status bit XXS7:	'Transfer request' (This bit will be reset if the transfer request is confirmed by the opposite side via the RCV status bit XRS6.
XXS6:	'Requested transfer direction' (only 'valid' when XXS7 is applied)
XXS5:	'Error detected' RCV status bit XRS6: A 'Transfer request' was recognized and accepted
XRS4:	The running data transfer was completed successfully
XRS3:	The running data transfer was completed 'by the destination'
XRS2...XRS0:	Various 'Error messages' in connection with XRS4

State Machine for a Transmitter/Receiver Module for High-Speed Serial Data Transmission: MACH110 and MACH210

Advanced Micro Devices

Application Note

by Günter Delfs, Deutsches Elektronen Synchotron, Germany

ALLGEMEINES ZUM HSSL-MODEL

HSSL –> High Speed Serial Link

Um hohe Datenraten über eine schnelle serielle Schnittstelle zu übertragen, wurde mit 'AMD TAXI chips' ein Modul entwickelt, das eine wahlweise Übertragung von Bytes bzw. 16-bit Worten erlaubt. Eine Übertragung von 32-bit Worten ist durch Erweiterung der Kontrollfunktion und der Logik ohne weiteres möglich. Anwendung findet dieses HSSL-Modul (High Speed Serial Link -) bei Verbindungen zwischen VME-Rechner (16-bzw. 32-bit Worte) und IBM(370)-Kanälen (Bytes).

Sie seriellen Ausgangs- bzw. Eingangssignale der TAXI chips werden über Lichtwellensender bzw. -empfänger geführt und ermöglichen damit eine getestete Lichtwellenleiterverbindung von mehr als 3.5 km Entfernung. Die Datenübertragungsrate wird durch die Taktfrequenz der TAXI chips und durch die Systemverwaltungszeiten bestimmt.

Anmerkung zur Funktionsbeschreibung des HSSL-Moduls

In der folgenden Kurzbeschreibung des HSSL-Moduls wird der Begriff 'Kontrollogik' des Senders/Empfängers verwendet. Hierzu gehören der MACH110/210, der PAL20RA10 (Figure 5-1) und der PALCE16V8 (Figure 5-1).

Im MACH110/210 wird der 'Read/Write-Byte/Word Transfer' mit der 'State Machine A' und die 'Requests' mit der 'State Machine B' kontrolliert.

Der PAL20RA10 und der PALCE16V8 kontrollieren die Einzelsignale der TAXI chips (DSTR, CSTR, RC0...2, VLIN usw.), Go/Stop-Requests, Errors usw. Das Design für den PAL20RA10 und PALCE16V8 wird hier nicht erklärt.

Das HSSL–Modul ist nur der Sende/Empfangsteil einer größeren Einheit (in der Beschreibung 'System' genannt). Die in der Beschreibung eingefügten STATES sollen das Lesen und Verstehen des MACH-Designs erleichtern.

Für den MACH110 muß folgendes beachtet werden: Da fast alle Eingangssignale asynchron sind, müssen sie für eine STATE MACHINE über ein vorgeschaltetes Register synchronisiert werden. Beim MACH210 wird das 'BURIED REGISTER' zur Synchronisation verwendet.

Beide Designs (MACH110/MACH210) liegen vor!

ERSTELLUNG DES DESIGNS FÜR DEN MACH110/210

Einige Ausgangssignale sollen aktiv 'low' sein. Dieses ist durch 'invert' für das entsprechende Ausgangssignal in 'declarations' anzugeben, z.B. mit

...... pin istype 'reg_d,invert';

Der 'Fitter' erzeugt jedoch dann mehr Produktterme für dieses Signal, als im MACH110/210 zugelassen sind. Ich habe diese Signale mit einem '_ ' gekennzeichnet und im Design in der Deklaration mit ...istype 'neg' angegeben.

KURZBESCHREIBUNG DES HSSL-MODULS:

Initialisierung des HSSL-Moduls

Das Sende-FIFO (XMT-FIFO) wird vor jeder Datenübertragung vom System mit einem Reset-Signal (XRXMT – Reset XMT_FIFO) gelöscht. Das Empfangs- FIFO (RCV-FIFO) wird vor jeder Datenübertragung durch ein vom Sender gesendetes Reset-RCV-Signal (RCVRS) gelöscht.

Der Sender gibt dazu das von seinem System gelieferte XRREM – Signal (Reset Remote) an die Kontrollogik (PAL20RA10), die ein 'send init' Request (RQRR) erzeugt.

Der RQRR hat gegenüber allen anderen Requests die höchste Priorität.

ñ state B02 ñ

(Refer to the design file for all state descriptions)

Von der Kontrollogik werden die Command-bits XCI0 und XCI1 mit dem dazugehörigen STRB (strobe) Signal über den XMT-TAXI chip an den Empfänger sendet.

ñ state B10 ñ

ñ state B11 ñ

Beim Empfänger werden diese beiden Command-bits als RCI0 und RCI1 erkannt (PALCE16V8) und daraufhin ein Reset-RCV-FIFO (RCVRS) vom PAL20RA10 gegeben.

Datenübertragung:

Daten können erst dann übertragen werden, wenn beim Sender und Empfänger von den Systemen der Kanal aufgesetzt worden ist und das Signal CHSL (channel select) an der Kontrollogik anliegt.

Die Übertragungsrichtung wird mit dem Signal CHRD (channel read) beim Aufsetzen des Kanals angegeben.

Sender XMT ñ state A01 ñ

Empfänger RCV ñ state A01 ñ

Mit dem Signal SW16 können die Daten wahlweise als Byte (XD0...7) oder als 16-bit Wort (XD0...15) übertragen werden.

Das Signal SW16 kann fest verdrahtet (switch) oder mit dem Aufsetzen des Kanals gesetzt werden.

Daten Senden:

Nachdem der Kanal aufgesetzt ist und Daten gesendet werden sollen, signalisiert die Kontrollogik dem System mit dem Signal XRDY (data input free for write), daß die Datenübertragung beginnen kann.

XMT ñ state A10 ñ

Die Sendedaten werden vom System mit dem Signal WDAT (write data) in das Sende-Latch (XMT-Latch) übergeben und gleichzeitig wird mit WDAT Signal der 'write-cycle' in der Kontrollogik gestartet.

Die Ausgänge des XMT-Latches werden mit den Signalen WEH (Write Enable High) bzw. WEL (Write Enable Low) Byte-weise freigegeben.

XMT ñ state A11 ñ

XMT ñ state A13 ñ

Mit dem dazugehörigen Signal WFI (Write into XMT-FIFO) werden die Daten ins XMT-FIFO geschrieben, sofern das FIFO mit seiner Full-Flag (FFR – Fifo free) signalisiert, daß es noch nicht 'voll' ist.

XMT ñ state A12 ñ

XMT ñ state A14 ñ

Die Empty-Flag des XMT-FIFOS gibt bereits nach dem ersten WFI ein 'send data'-Request (RQDT) an die Kontrollogik,

XMT ñ state B02 ñ

die dann mit dem XRD Signal (Read XMT-FIFO) die Daten aus dem FIFO ausliest und sie mit dem dazugehörigen Strobe-Signal STRB an den XMT-TAXI chip übergibt.

XMT ñ state B50 ñ

XMT ñ state B51 ñ

Mit dem Signal ACK bestätigt der XMT-TAXI chip die Datenübernahme.Weitere Daten werden erst wieder an den XMT-TAXI chip übergeben, wenn ACK wieder zurückgenommen wurde.

XMT ñ state B02 ñ

Wird das letzte Byte bzw. 16-bit Wort vom Sytem übertragen, liegt das Signal XEOR (end of write data) mit den Daten-bits am Eingang des XMT-FIFOS und wird mit den Daten übertragen.

Das Signal XEOR ist das vom System erzeugte Signal XXEOR. Es wird bis zum Abschalten des Kanals und damit des Signals CHSL gehalten.

Ist die Datenübertragung abgeschlossen, erhält der Sender über die Status-bits eine 'Transfer End' Rückmeldung vom Empfänger. Das System des Senders schaltet daraufhin den Kanal ab und das Signal CHSL verschwindet. Die Kontrollogik geht somit wieder in die Warteschleife ('idle loop') und wartet auf das nächste CHSL.

XMT ñ state A01 ñ

Daten Empfangen:

Die Empfangsdaten werden vom Ausgang des RCV-TAXI chips mit dem Signal WDST (write received data into FIFO) in das Empfangs-FIFO (RCV-FIFO) Übernommen. Die Empty-Flag des RCV-FIFOS signalisiert daraufhin mit dem Signal FDA (fifo data available) der Kontrollogik, daß Daten anstehen.

RCV ñ state A20 ñ

Werden vom System keine Daten aus dem Empfangs-Latch (RCV-Latch) ausgelesen und das Signal RDT (RCV-FIFO data read out done) liegt somit nicht an, startet die Kontrollogik den 'read-cycle'.

RCV ñ state A20 ñ

Die Eingänge vom RCV-Latch werden mit den Signalen REH (read enable high) bzw. REL (read enable low) Byte-weise geöffnet und gleichzeitig werden mit dem Signal RFI (read data from RCV-FIFO) die Daten aus dem RCV-FIFO in das RCV-Latch übernommen.

RCV ñ state A21 ñ

RCV ñ state A22 ñ

RCV ñ state A23 ñ

Sind die Daten im RCV-Latch, signalisiert die Kontrollogik dem System mit dem Signal XRDY (read data available), daß die Daten übernommen werden können.

RCV ñ state A24 ñ

Die Kontrollogik wartet nun solange, bis das System mit dem Signal RDT (RCV -FIFO data read out done) die Datenübernahme bestätigt. Erst dann werden weitere Daten aus dem RCV-FIFO gelesen.

RCV ñ state A24 ñ

RCV ñ state A20 ñ

Wird mit den Daten auch das Signal REOR (read end of record) aus dem RCV-FIFO ausgelesen, dann gibt die

Kontrollogik das Signal XEOR (end of read data) an das System.

Die Kontrollogik wartet nun darauf, daß das Signal CHSL weggenommen wird und geht erst dann wieder in die Warteschleife ('idle loop').

RCV ñ state A30 ñ

RCV ñ state A01 ñ

Status:

Für die Status-Bytes Übertragung muß der Kanal nicht aufgesetzt werden. Es wird ein 8-bit Status gesendet bzw. empfangen.

Die Status Übertragung hat die zweithöchste Priorität. Das Status Senden wird durch Änderung eines oder mehrerer Status-bits (XXS 0...7) ausgelöst. Ein 8-bit Comparator stellt die Änderung fest und gibt einen 'send status'-Request (RQSTA) an die Kontrollogik.

ñ state B02 ñ

Steht kein Request mit höherer Priorität an (z.B. RQRR), dann wird der RQSTA ausgeführt.

ñ state B20 ñ

Das Signal LST (latch xmitter status) übergibt das veränderte Status-Byte an das XMT-STATUS Register und an den Comparator, der damit den RQSTA wieder wegnimmt.

Mit der Übergabe an den XMT-TAXI chip durch das STRB Signal wird ein zusätzliches Command-bit (XCI2) gesendet, damit auf der Empfangsseite die Status-bits nicht im RCV-FIFO sondern im Empfangsstatusregister RCV- STATUS (XRS 0...7) übernommen werden.

ñ state B21 ñ

Data Flow Control

Das RCV-FIFO ist relativ klein (1K * 9 Bit).

Werden in das RCV-FIFO Daten schneller geschrieben als gelesen, dann wird bei halbvollem RCV-FIFO die Half-Full-Flag RHF (RHF -> RCV Half Full Flag) gesetzt. Damit wird der Kontrollogik signalisiert, daß keine Daten mehr empfangen werden können.

Ein 'send xoff' Request RQSTP (STOP Request) wird daraufhin erzeugt, der einen Command-bit Transfer zum Sender hin bewirkt.

RCV ñ state B02 ñ

Mit dem Command-bit XCI0 und dem Strobe Signal STRB wird dieser STOP-Befehl über den XMT-TAXI chip des Datenempfängers an den Datensender gegeben.

RCV ñ state B30 ñ

RCV ñ state B31 ñ

Der Sender erkennt an dem empfangenem Command-bit RCI0, daß keine Daten mehr gesendet werden sollen. Die Kontrollogik des Senders (PALCE16V8) erzeugt daraufhin ein STOPX Signal, das einen weiteren Datentransfer verhindert.

XMT ñ state B02 ñ

Hat nun der Datenempfänger sein RCV-FIFO weiter ausgelesen und ist dessen Half-Full-Flag zurückgesetzt worden, dann wird ein 'send xon' Request RQGO (GO Request) an die Kontrollogik gegeben.

RCV ñ state B02 ñ

Mit dem Command-bit XCI1 und dem Strobe Signal STRB wird dieser GO-Befehl über den XMT-TAXI chip des Datenempfängers an den Datensender gegeben.

RCV ñ state B40 ñ

RCV ñ state B41 ñ

Der Sender erkennt an dem empfangenem Command-bit RCI1, daß Daten wieder gesendet werden können.

Die Kontrollogik des Senders nimmt daraufhin das STOPX Signal zurück. Das Request RQDT wird freigegeben und Daten können übertragen werden.

RCV ñ state B02 ñ

RCV ñ state B50 ñ

RCV ñ state B51 ñ

Kommunikation zwischen Sender und Empfänger

Es sollen hier nur die wichtigen Status-bit-Bedeutungen erläutert werden.

XMT-Status-bit XXS7:	'Transfer request' (Dieses bit wird zurückgesetzt, wenn über das RCV-Status-bit XRS6 der Transfer Request von der Gegenseite bestätigt wird.
XXS6:	'Requested transfer direction' (nur 'valid' wenn XXS7 ansteht)
XXS5:	'Error detected'
RCV-Status-bit XRS6:	Ein 'Transfer request' wurde erkannt und akzeptiert.
XRS4:	Der laufende Datentransfer wurde erfolgreich beendet
XRS3:	Der laufende Datentransfer wurde beendet 'by the destination'
XRS2...XRS0:	Verschiedene 'Error-messages' in Verbindung mit XRS4

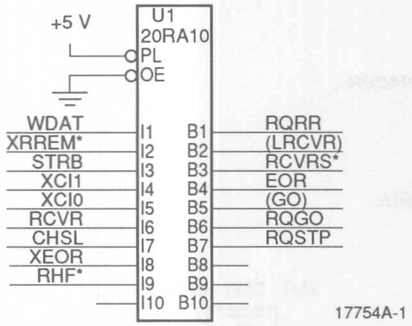


Figure 5-1. PAL20RA10

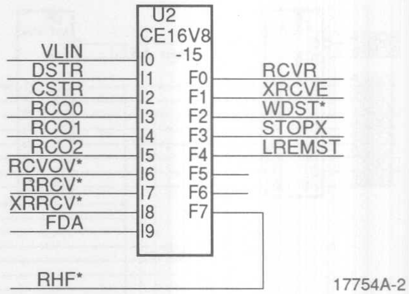


Figure 5-2. PALCE16V8

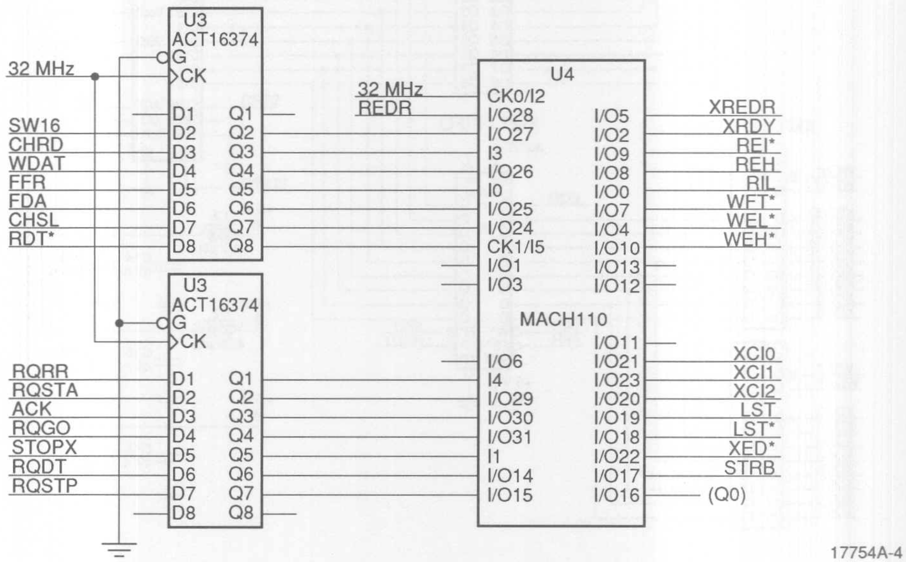


Figure 5-3. HSSL State Machine

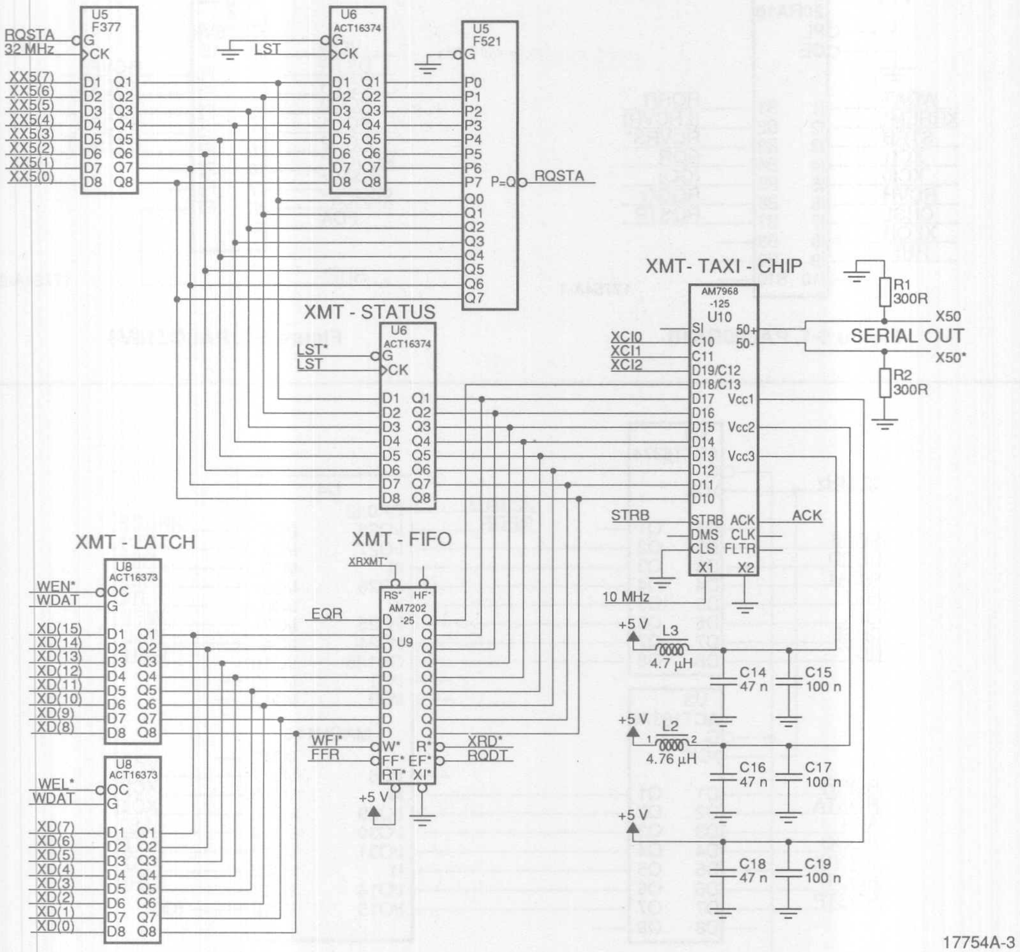


Figure 5-4. HSSL Transmitter

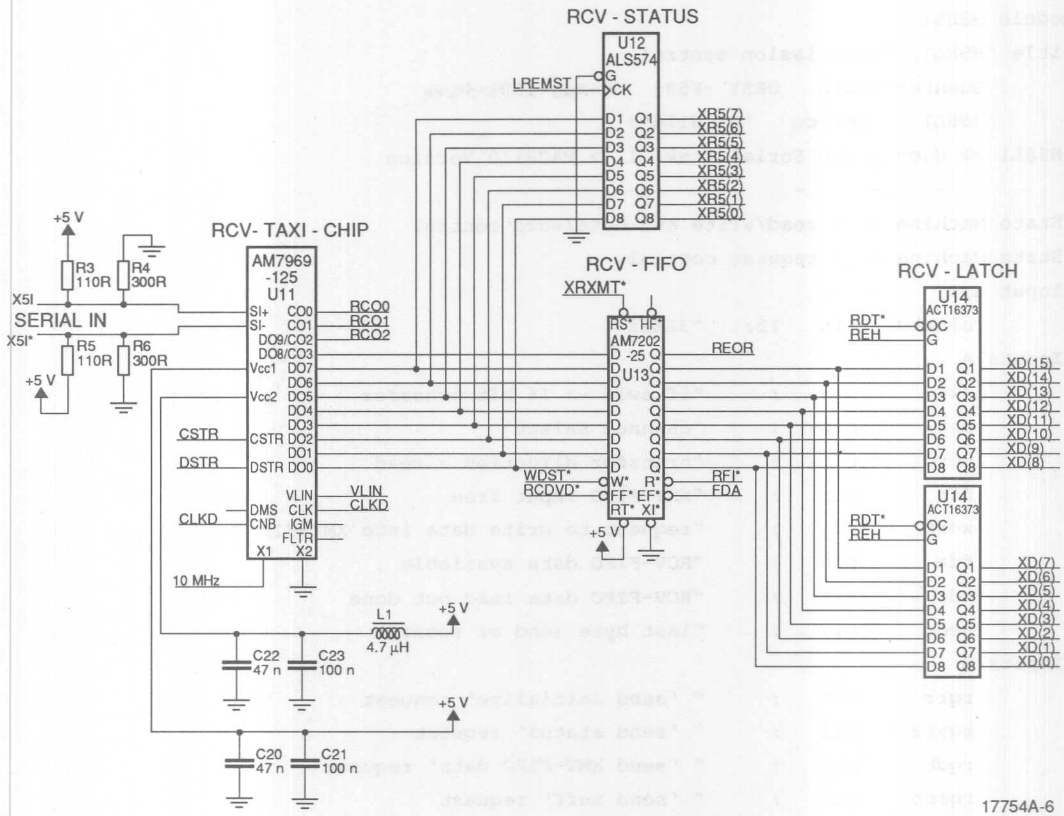


Figure 5-5. HSSL Receiver

DESIGN FILE: MACH110

```
module HSSL1
title 'HSSL1, Transmission control
      Guenter Delfs   DESY -F58-   5-Aug-1991 '
      HSSL1    device   'MACH110';
"HSSL1 -> High Speed Serial Link  1 -> MACH110 Version
"          -    -     -      -     -
"State Machine A -> read/write and byte/word control
"State Machine B -> request control
"Input A,B
       clock0   pin   13;   "32MHz
"Inputs A
       sw16     pin   ;     "if sw16 -> 16 bit transfer
       chsl     pin   ;     "channel select
       chrd     pin   ;     "transfer direction = read
       ffr      pin   ;     "XMT-FIFO input free
       wdat     pin   ;     "request to write data into XMT-FIFO
       fda      pin   ;     "RCV-FIFO data available
      !rdt_     pin   ;     "RCV-FIFO data read out done
       reor     pin   ;     "last byte (end of record)
"Inputs B
       rqrr     pin   ;     " 'send initialize' request
       rqsta    pin   ;     " 'send status' request
       rqdt     pin   ;     " 'send XMT-FIFO data' request
       rqstp    pin   ;     " 'send xoff' request
       rqgo     pin   ;     " 'send xon' request
       stopx    pin   ;     "stop FIFO data transfer
       ack      pin   ;     "TAXI input strobe received
"Outputs A
       wel_     pin   istype 'reg,neg'; "write low (second) byte enable
                                        "(perm. if byte transfer)
       weh_     pin   istype 'reg,neg'; "write high (first) byte enable
       wfi_     pin   istype 'reg,neg'; "write into XMT-FIFO
       rfi_     pin   istype 'reg,neg'; "read from RCV-FIFO
       rel      pin   istype 'reg';     "read low (second) byte from
                                        "RCV-FIFO
       reh      pin   istype 'reg';     "read high (first) byte from
                                        "RCV-FIFO (perm. if byte transfer)
       xrdy     pin   istype 'reg';     "data available (read) or
                                        "data input free (write)
       xreor    pin   istype 'reg';     "end of read data
"Outputs B
       xci0     pin   istype 'reg,';     "xmitter command bit
```

```
        xci1     pin   istype 'reg,';      "xmitter command bit
        xci2     pin   istype 'reg,';      "xmitter command bit
        lst      pin   istype 'reg,';      "latch xmitter status
        lst_     pin   istype 'reg,neg';
        xrd_     pin   istype 'reg,neg';   "read data from XMT-FIFO
        strb     pin   istype 'reg,';      "strobe data into xmitter
        q0       pin   istype 'reg,';
        sregA  =   [wel_,weh_,wfi_,rfi_,reh,rel,xrdy,xreor];
        sregB  =   [xci0,xci1,xci2,lst,lst_,xrd_,strb,q0];
"State Values...A
"================
        A01 =[0,0,0,0,0,0,0,0];
"rel_ (A10...A13) used for significance only
        A10 =[0,0,0,0,0,1,1,0]; "xrdy,(rel_)
        A11 =[0,1,0,0,0,1,0,0]; "weh_,(rel_)
        A12 =[0,1,1,0,0,0,0,0]; "weh_,wfi_
        A13 =[1,0,0,0,0,1,0,0]; "wel_,(rel_)
        A14 =[1,0,1,0,0,0,0,0]; "wel_,wfi_
"wel_,weh_ (A20...A22) used for significance only
        A20 =[1,0,0,0,0,0,0,0]; "(wel_)
        A21 =[0,0,0,1,1,0,0,0]; "reh,rfi_
        A22 =[0,1,0,0,0,0,0,0]; "(weh_)
        A23 =[0,0,0,1,0,1,0,0]; "rel,rfi_
        A24 =[0,0,0,0,0,0,1,0]; "xrdy
        A30 =[0,0,0,0,0,0,0,1]; "xreor
"State Values...B
"================
        B01 = [0,0,0,0,0,0,0,1];
        B02 = [0,0,0,0,0,0,0,0];
        B10 = [1,1,0,0,0,0,0,0]; "xci0,xci1
        B11 = [1,1,0,0,0,0,1,0]; "xci0,xci1,strb
        B20 = [0,0,1,1,1,0,0,0]; "xci2,lst,lst_
        B21 = [0,0,1,0,1,0,1,0]; "xci2,lst_,strb

        B30 = [1,0,0,0,0,0,0,0]; "xci0
        B31 = [1,0,0,0,0,0,1,0]; "xci0,strb
        B40 = [0,1,0,0,0,0,0,0]; "xci1
        B41 = [0,1,0,0,0,0,1,0]; "xci1,strb
        B50 = [0,0,0,0,0,1,0,0]; "xrd_
        B51 = [0,0,0,0,0,1,1,0]; "xrd_,strb
equations
       [wel_,weh_,wfi_,rfi_,reh,rel,xrdy,xreor].clk = clock0;
       [xci0,xci1,xci2,lst,lst_,xrd_,strb,q0].clk   = clock0;
```

```
state_diagram sregA;
"idle loop
"=========
      state A01:
        IF (!chsl) THEN A01;                    "wait for chsl
        IF (chsl & !chrd) THEN A10;             "chsl,write cycle
        IF (chsl & chrd) THEN A20;              "chsl,read cycle
"write cycle
"===========
      state A10:"xrdy
        IF (chsl & !wdat) THEN A10;             "wait for wdat
        IF (chsl & wdat & !sw16) THEN A13;      "byte write cycle
        IF (chsl & wdat &  sw16) THEN A11;      "word write cycle
        IF (!chsl) THEN A01;                    "!chsl,forced return
      state A11:"weh_
        IF (chsl & !ffr) THEN A11;              "wait for ffr
        IF (chsl & ffr) THEN A12;
        IF (!chsl) THEN A01;                    "!chsl,forced return
      state A12:"weh_,wfi_
        GOTO A13;
      state A13:"wel_
        IF (chsl & !ffr) THEN A13;              "wait for ffr
        IF (chsl & ffr) THEN A14;
        IF (!chsl) THEN A01;                    "!chsl,forced return
      state A14:"wel_,wfi_
        GOTO A10;
"read cycle
"==========
      state A20:
        IF (chsl & !fda) THEN A20;             "wait for fda
        IF (chsl & fda & rdt_) THEN A20;       "wait for !rdt_
        IF (chsl & fda & !rdt_) THEN A21;
        IF (!chsl) THEN A01;                   "!chsl,forced return
      state A21:"reh_,rfi_
        IF (!sw16 & !reor) THEN A24;           "byte read cycle
        IF (sw16 & !reor) THEN A22;            "word read cycle
        IF (reor) THEN A30;                    "end of record
     state A22:
        IF (chsl & !fda) THEN A22;             "wait for fda
        IF (chsl & fda) THEN A23;
        IF (!chsl) THEN A01;                   "!chsl,forced return
      state A23:"rel_,rfi_
        IF (!reor) THEN A24;
```

```
      IF (reor) THEN A30;                "end of record
    state A24:"xrdy
      IF (chsl & !rdt_) THEN A24;        "wait for rdt_
      IF (chsl & rdt_) THEN A20;
      IF (!chsl) THEN A01;               "!chsl,forced return
    state A30:"xreor
      IF (chsl) THEN A30;                "wait for !chsl
      IF (!chsl) THEN A01;               "!chsl,forced return
state_diagram sregB;
"wait for request
"================
    state B01:
      GOTO B02;                          "delay cycle
    state B02:"request handling
      IF (ack) THEN B02;
      IF (!rqrr & !rqsta & !rqdt & !rqstp & !rqgo) THEN B02;
      IF (rqrr & !ack) THEN B10;
      IF (!rqrr & rqsta & !ack) THEN B20;
      IF (!rqrr & !rqsta & rqstp & !ack) THEN B30;
      IF (!rqrr & !rqsta & !rqstp & rqgo & !ack) THEN B40;
      IF (!rqrr & !rqsta & rqdt & !rqstp & !rqgo & stopx &!ack) THEN B02;
      IF (!rqrr & !rqsta & rqdt & !rqstp & !rqgo & !stopx &!ack) THEN B50;
" 'init' request
"===============
    state B10:"xci0,xci1
      GOTO B11;
    state B11:"xci0,xci1,strb
      GOTO B01;
" 'status' request
"=================
    state B20:"xci2,lst,lst_
      GOTO B21;
    state B21:"xci2,lst_,strb
      GOTO B01;
" 'xoff' request
"===============
    state B30:"xci0
      GOTO B31;
    state B31:"xci0,strb
      GOTO B01;
" 'xon' request
"==============
    state B40:"xci1
```

```
      GOTO B41;
    state B41:"xci1,strb
      GOTO B01;
" 'data' request
"===============
    state B50:"xrd_
      GOTO B51;
    state B51:"xrd_,strb
      GOTO B02;
test_vectors "read/write - byte/word control
([clock0,sw16,chsl,chrd,ffr,wdat,fda,rdt_,reor]->[sregA])
 [ .c. ,  0 ,  0  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
 [ .c. ,  0 ,  0  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
 [ .c. ,  0 ,  1  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
 [ .c. ,  0 ,  1  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
 [ .c. ,  0 ,  1  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A13 ];
 [ .c. ,  0 ,  1  ,  0 , 0 ,  1 , 0 , 0  ,  0 ]->[ A13 ];
 [ .c. ,  0 ,  1  ,  0 , 1 ,  1 , 0 , 0  ,  0 ]->[ A14 ];
 [ .c. ,  0 ,  1  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
 [ .c. ,  0 ,  1  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
 [ .c. ,  0 ,  0  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
 [ .c. ,  0 ,  0  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
"------------------------------------------------------------
" [ .c. ,  0 ,  0  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
" [ .c. ,  1 ,  1  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
" [ .c. ,  1 ,  1  ,  0 , 0 ,  1 , 0 , 0  ,  0 ]->[ A11 ];
" [ .c. ,  1 ,  1  ,  0 , 0 ,  1 , 0 , 0  ,  0 ]->[ A11 ];
" [ .c. ,  1 ,  1  ,  0 , 1 ,  1 , 0 , 0  ,  0 ]->[ A12 ];
" [ .c. ,  1 ,  1  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A13 ];
" [ .c. ,  1 ,  1  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A14 ];
" [ .c. ,  1 ,  1  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
" [ .c. ,  1 ,  1  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A10 ];
" [ .c. ,  1 ,  0  ,  0 , 1 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
"------------------------------------------------------------
"[clock0,sw16,chsl,chrd,ffr,wdat,fda,rdt_,reor ]->[sregA]
" [ .c. ,  0 ,  0  ,  0 , 0 ,  0 , 0 , 0  ,  0 ]->[ A01 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 0 , 0  ,  0 ]->[ A20 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 0 , 1  ,  0 ]->[ A20 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 1 , 1  ,  0 ]->[ A20 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 1 , 1  ,  0 ]->[ A20 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 1 , 0  ,  0 ]->[ A21 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 1 , 0  ,  0 ]->[ A24 ];
" [ .c. ,  0 ,  1  ,  1 , 0 ,  0 , 1 , 0  ,  0 ]->[ A24 ];
```

```
" [ .c. , 0 , 1  , 1 , 0 , 0 , 1 , 1  , 0 ]->[ A20 ];
" [ .c. , 0 , 1  , 1 , 0 , 0 , 1 , 1  , 0 ]->[ A20 ];
" [ .c. , 0 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A21 ];
" [ .c. , 0 , 1  , 1 , 0 , 0 , 1 , 0  , 1 ]->[ A30 ];
" [ .c. , 0 , 1  , 1 , 0 , 0 , 1 , 0  , 1 ]->[ A30 ];
" [ .c. , 0 , 0  , 0 , 0 , 0 , 0 , 0  , 0 ]->[ A01 ];
"--------------------------------------------------------
" [ .c. , 0 , 0  , 0 , 0 , 0 , 1 , 0  , 0 ]->[ A01 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 1  , 0 ]->[ A20 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 1  , 0 ]->[ A20 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A21 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 0 , 0  , 0 ]->[ A22 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 0 , 0  , 0 ]->[ A22 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A23 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A24 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A24 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 1  , 0 ]->[ A20 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A21 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A22 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 0 ]->[ A23 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 1 ]->[ A30 ];
" [ .c. , 1 , 1  , 1 , 0 , 0 , 1 , 0  , 1 ]->[ A30 ];
" [ .c. , 1 , 0  , 1 , 0 , 0 , 1 , 0  , 1 ]->[ A01 ];
" [ .c. , 1 , 0  , 0 , 0 , 0 , 0 , 0  , 0 ]->[ A01 ];
"===========================================================
"test_vectors  request control
"([clock0,rqrr,rqsta,rqdt,rqstp,rqgo,stopx,ack]->[sregB])
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B02 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 1 ]->[ B02 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B02 ];
" [ .c. , 1  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B10 ];
" [ .c. , 1  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B11 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B01 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B02 ];
" [ .c. , 0  , 1  , 1 , 1  , 1  , 0  , 0 ]->[ B20 ];
" [ .c. , 0  , 1  , 1 , 1  , 1  , 0  , 0 ]->[ B21 ];
" [ .c. , 0  , 0  , 1 , 1  , 1  , 0  , 0 ]->[ B01 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B02 ];
" [ .c. , 0  , 0  , 0 , 1  , 0  , 0  , 0 ]->[ B30 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B31 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B01 ];
" [ .c. , 0  , 0  , 0 , 0  , 0  , 0  , 0 ]->[ B02 ];
" [ .c. , 0  , 0  , 0 , 0  , 1  , 0  , 0 ]->[ B40 ];
```

```
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B41 ];
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B01 ];
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
" [ .c. , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ B50 ];
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B51 ];
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
end
```

FITTER FILE: MACH110

Note: This file has been condensed in order to save trees.

```
AMD MACH FITR - MARKET RELEASE (1-24-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990
Flags Used:              Unplace=False          Max Packing=True
Flags Used:          Expand Small=False          Expand All=False
*********************************************************************
     Mach PLD Fitter - v 1.46          HSSL1, Transmission control
*********************************************************************
*** Timing Analysis for Signals
 Parameter    Min    Max          Signal List (Those having Max delay.)
      Tsu      1      1              wel_            weh_            xrdy
                                     wfi_            rfi_             rel
                                      reh            xrd_
      Tco      0      0              wel_            weh_            wfi_
                                     rfi_             reh             rel
                                     xrdy             q0
      Tcr      1      1              weh_            wfi_            rfi_
                                      rel             reh            xrdy
                                    xreor            strb
.
 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Register clock to combinatorial output
 Tcr - Register thru combinatorial logic to setup
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                Available         Used          Remaining
       Clocks:      2               1              1
         Pins:     38              32              6      ->     84%
    I/O Macro:     32              16             16
  Total Macro:     32              16             16
 Product Terms:   128              52             52      ->     58%

MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
*** Last Equations Placed in Blocks
Weakly -
Assign -
*** Block Partitioning Results
                 Array     Macros      # I/O     Buried   Product     Signal
                Inputs     Remain      Macro      Logic     Terms     Fanout
   Block-> A      16         8           8          0        44          8
   Block-> B      15         8           8          0        32          8
*** Block Signal List
Block-> A             xreor            rel             wfi_            reh
                       rfi_            weh_            xrdy            wel_
Block-> B                q0            strb            lst_            lst
                       xci2            xci0            xrd_            xci1
ö> INFORMATION F050 - Device Utilization....... *: 70 %
```

```
*** Feedback Map - HSSL1, Transmission control

 Gbl Inp .--.      I/O   .--+--A--+--.  I/O            I/O  .--+--B--+--.  I/O
         | 0|      rel : 0|      |21|                  q0 : 0|      |21| rqrr
         | 1|          | 1|      |20| chrd           strb : 1|      |20|
         | 2|     xrdy : 2|      |19|                lst_ : 2|      |19|
         | 3|          | 3|      |18| rdt_            lst : 3|      |18|
         | 4|     wel_ : 4|      |17|                xci2 : 4|      |17| stopx
         | 5|    xreor : 5|      |16| ffr            xci0 : 5|      |16|
         '--'          | 6|      |15: chsl           xrd_ : 6|      |15|
                  wfi_ : 7|      |14: fda            xci1 : 7|      |14|
                   reh : 8|      |13: wdat           rqgo : 8|      |13: rqdt
                  rfi_ : 9|      |12: sw16            ack : 9|      |12: rqstp
                  weh_ :10|      |11: reor          rqsta :10|      |11|
                       '--+-u--u+--'                       '--+-u--u+--'

*** Logic Map - HSSL1, Transmission control

 Gbl Inp .--.      I/O   .--+--A--+--.  I/O            I/O  .--+--B--+--.  I/O
      ffr| 0|      rel | 0| 7    |21|                  q0 | 0| 3    |21|
    stopx| 1|          | 1| *    |20|                strb | 1| 4    |20|
   clock0| 2|     xrdy | 2| 6    |19|                lst_ | 2| 2    |19|
     chrd| 3|          | 3| *    |18|                 lst | 3| 1    |18|
     rqrr| 4|     wel_ | 4| 7    |17|                xci2 | 4| 2    |17|
     rdt_| 5|    xreor | 5| 3    |16|                xci0 | 5| 3    |16|
         '--'          | 6| *   .|15|                xrd_ | 6| 2   .|15|
                  wfi_ | 7| 2   .|14|                xci1 | 7| 3   .|14|
                   reh | 8| 1   .|13|                     | 8| .   .|13|
                  rfi_ | 9| 2   .|12|                     | 9| .   .|12|
                  weh_ |10| 4   .|11|                     |10| .   .|11|
                       '--+-u--u+--'                       '--+-u--u+--'
```

```
*** Pin Map - HSSL1, Transmission control


                                    rel
                                     . |       rqgo
                             xrdy   |  |        |  ack
                              .  |  |  |        |  | rqsta
                      wel_ |  |  |  |  |        |  |  | reor
                           |  |  |  |  |        |  |  |  |
                     .-----'--'--'--'--'--o-----'--'--'--'---.
                     |                       4  4  4  4  4   |
                     |     6  5  4  3  2  1  4  3  2  1  0   |
                xreor| 7                                   39|sw16
                     | 8                  G  V             38|wdat
                 wfi_| 9                  n  c             37|fda
                  ffr|10                  d  c             36|chsl
                stopx|11                                   35|rdt_
                 Gnd |12           MACH-110                34| Gnd
               clock0|13                                   33|rqrr
                  reh|14             V  G                  32|chrd
                 rfi_|15             c  n                  31|xci1
                 weh_|16             c  d                  30|xrd_
                     |17                                   29|xci0
                     |   1  1  2  2  2  2  2  2  2  2  2     |
                     |   8  9  0  1  2  3  4  5  6  7  8     |
                     '---.--.--.--.--------.--.--.--.--.-----'
                         |  |  |  |        |  |  |  |  |
                         '  |  |  |        |  |  |  |  xci2
                            '  |  |        |  |  | lst
                           rqdt   |        |  | lst_
                             rqstp         | strb
                                          q0
The Design Doc is  stored in ===> hssl1.Rpt
The Jedec Data is  stored in ===> hssl1.Jed
The Placements are stored in ===> hssl1.Plc
The  Fuse Plot is  stored in ===> hssl1.Xpt

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: hssl1
```

PLACEMENT FILE: MACH110

```
; Flags Used:           Unplace=False         Max Packing=True
; Flags Used:      Expand Small=False          Expand All=False

; FITR generated placements

Pin    13           clock0  ; Inp        ; I  2
Pin    39             sw16  ; Inp        ; B 11
Pin    36             chsl  ; Inp        ; B  8
Pin    32             chrd  ; Inp        ; I  3
Pin    10              ffr  ; Inp        ; I  0
Pin    38             wdat  ; Inp        ; B 10
Pin    37              fda  ; Inp        ; B  9
Pin    35             rdt_  ; Inp        ; I  5
Pin    40             reor  ; Inp        ; B 12
Pin    33             rqrr  ; Inp        ; I  4
Pin    41            rqsta  ; Inp        ; B 13
Pin    20             rqdt  ; Inp        ; A 14
Pin    21            rqstp  ; Inp        ; A 15
Pin    43             rqgo  ; Inp        ; B 15
Pin    11            stopx  ; Inp        ; I  1
Pin    42              ack  ; Inp        ; B 14
Node    6           wel__I    Reg        ; A  4
Node   12           weh__I    Reg        ; A 10
Node    9           wfi__I    Reg        ; A  7
Node   11           rfi__I    Reg        ; A  9
Node    2            rel_I    Reg        ; A  0
Node   10            reh_I    Reg        ; A  8
Node    4           xrdy_I    Reg        ; A  2
Node    7          xreor_I    Reg        ; A  5
Node   23           xci0_I    Reg        ; B  5
Node   25           xci1_I    Reg        ; B  7
Node   22           xci2_I    Reg        ; B  4
Node   21            lst_I    Reg        ; B  3
Node   20           lst__I    Reg        ; B  2
Node   24           xrd__I    Reg        ; B  6
Node   19           strb_I    Reg        ; B  1
Node   18             q0_I    Reg        ; B  0
Pin     6             wel_    Reg        ; A  4
Pin    16             weh_    Reg        ; A 10
Pin     9             wfi_    Reg        ; A  7
Pin    15             rfi_    Reg        ; A  9
Pin    14              reh    Reg        ; A  8
Pin     2              rel    Reg        ; A  0
Pin     4             xrdy    Reg        ; A  2
Pin     7            xreor    Reg        ; A  5
Pin    29             xci0    Reg        ; B  5
Pin    31             xci1    Reg        ; B  7
Pin    28             xci2    Reg        ; B  4
Pin    27              lst    Reg        ; B  3
Pin    26             lst_    Reg        ; B  2
Pin    30             xrd_    Reg        ; B  6
Pin    25             strb    Reg        ; B  1
Pin    24               q0    Reg        ; B  0

; Group Mach_Seg_A xreor rel wfi_ reh rfi_ weh_ xrdy wel_
; Group Mach_Seg_B q0 strb lst_ lst xci2 xci0 xrd_ xci1
```

DESIGN FILE: MACH210

```
module HSSL2
title 'HSSL2, Transmission and byte/word control
       Guenter Delfs   DESY -F58-   5-Aug-1991 '
       HSSL2     device   'MACH210';

"HSSL2 -> High Speed Serial Link  2 -> MACH210 Version
"         -      -     -     -     -

"State Machine A -> read/write and byte/word control
"State Machine B -> request control

"Input A,B
        clock0   pin   13;   "32MHz
"Inputs A
        sw16     pin   ;     "16 bit transfer
        chsl     pin   ;     "channel select
        chrd     pin   ;     "transfer direction = read
        ffr      pin   ;     "XMT-FIFO input free
        wdat     pin   ;     "request to write data into XMT-FIFO
        fda      pin   ;     "RCV-FIFO data available
       !rdt_     pin   ;     "RCV-FIFO data read out done
        reor     pin   ;     "last byte (end of record)

        bsw16    node  istype 'reg_d,buffer';
        bchsl    node  istype 'reg_d,buffer';
        bchrd    node  istype 'reg_d,buffer';
        bffr     node  istype 'reg_d,buffer';
        bwdat    node  istype 'reg_d,buffer';
        bfda     node  istype 'reg_d,buffer';
        brdt_    node  istype 'reg_d,buffer';
"Inputs B
        rqrr     pin   ;     " 'send initialize' request
        rqsta    pin   ;     " 'send status' request
        rqdt     pin   ;     " 'send XMT-FIFO data' request
        rqstp    pin   ;     " 'send xoff' request
        rqgo     pin   ;     " 'send xon' request
        stopx    pin   ;     "stop FIFO data transfer
        ack      pin   ;     "TAXI input strobe received

        brqrr    node  istype 'reg_d,buffer';
        brqsta   node  istype 'reg_d,buffer';
        brqdt    node  istype 'reg_d,buffer';
        brqstp   node  istype 'reg_d,buffer';
        brqgo    node  istype 'reg_d,buffer';
        bstopx   node  istype 'reg_d,buffer';
        back     node  istype 'reg_d,buffer';
"Outputs A
        wel_     pin   istype 'reg,neg'; "write low (second) byte enable
                                         "(perm. if byte transfer)
        weh_     pin   istype 'reg,neg'; "write high (first) byte enable
        wfi_     pin   istype 'reg,neg'; "write into XMT-FIFO
        rfi_     pin   istype 'reg,neg'; "read from RCV-FIFO
        rel      pin   istype 'reg,';    "read low (second) byte from
                                         "RCV-FIFO
        reh      pin   istype 'reg,';    "read high (first) byte from
                                         "RCV-FIFO (perm. if byte transfer)
        xrdy     pin   istype 'reg,';    "data available (read) or
                                         "data input free (write)
        xreor     pin   istype 'reg,';    "end of read data
"Outputs B
        xci0     pin   istype 'reg,';     "xmitter command bit
        xci1     pin   istype 'reg,';     "xmitter command bit
```

```
    xci2     pin   istype 'reg,';      "xmitter command bit
    lst      pin   istype 'reg,';      "latch xmitter status
    lst_     pin   istype 'reg,neg';
    xrd_     pin   istype 'reg,neg';   "read data from XMT-FIFO
    strb     pin   istype 'reg,';      "strobe data into xmitter
    q0       pin   istype 'reg,';

    sregA  =   [wel_,weh_,wfi_,rfi_,reh,rel,xrdy,xreor];

    sregB  =   [xci0,xci1,xci2,lst,lst_,xrd_,strb,q0];

"State Values...A
"================
    A01 = [0,0,0,0,0,0,0,0];

"rel_ (A10...A13) used for significance only
    A10 = [0,0,0,0,0,1,1,0]; "xrdy,(rel_)
    A11 = [0,1,0,0,0,1,0,0]; "weh_,(rel_)
    A12 = [0,1,1,0,0,0,0,0]; "weh_,wfi_
    A13 = [1,0,0,0,0,1,0,0]; "wel_,(rel_)
    A14 = [1,0,1,0,0,0,0,0]; "wel_,wfi_

"wel_,weh_ (A20...A22) used for significance only
    A20 = [1,0,0,0,0,0,0,0]; "(wel_)
    A21 = [0,0,0,1,1,0,0,0]; "reh,rfi_
    A22 = [0,1,0,0,0,0,0,0]; "(weh_)
    A23 = [0,0,0,1,0,1,0,0]; "rel,rfi_
    A24 = [0,0,0,0,0,0,1,0]; "xrdy

    A30 = [0,0,0,0,0,0,0,1]; "xreor

"State Values...B
"================
    B01 = [0,0,0,0,0,0,0,1];
    B02 = [0,0,0,0,0,0,0,0];

    B10 = [1,1,0,0,0,0,0,0]; "xci0,xci1
    B11 = [1,1,0,0,0,0,1,0]; "xci0,xci1,strb

    B20 = [0,0,1,1,1,0,0,0]; "xci2,lst,lst_
    B21 = [0,0,1,0,1,0,1,0]; "xci2,lst_,strb

    B30 = [1,0,0,0,0,0,0,0]; "xci0
    B31 = [1,0,0,0,0,0,1,0]; "xci0,strb

    B40 = [0,1,0,0,0,0,0,0]; "xci1
    B41 = [0,1,0,0,0,0,1,0]; "xci1,strb

    B50 = [0,0,0,0,0,1,0,0]; "xrd_
    B51 = [0,0,0,0,0,1,1,0]; "xrd_,strb

equations

    bsw16        := sw16;
    bsw16.clk    = clock0;

    bchsl        := chsl;
    bchsl.clk    = clock0;

    bchrd        := chrd;
    bchrd.clk    = clock0;

    bffr         := ffr;
    bffr.clk     = clock0;

    bwdat        := wdat;
    bwdat.clk    = clock0;

    bfda         := fda;
    bfda.clk     = clock0;
```

```
        brdt_      := rdt_;
        brdt_.clk   = clock0;

        brqrr      := rqrr;
        brqrr.clk   = clock0;

        brqsta     := rqsta;
        brqsta.clk = clock0;

        brqdt      := rqdt;
        brqdt.clk   = clock0;

        brqstp     := rqstp;
        brqstp.clk = clock0;

        brqgo      := rqgo;
        brqgo.clk   = clock0;

        bstopx     := stopx;
        bstopx.clk = clock0;

        back       := ack;
        back.clk    = clock0;

        [wel_,weh_,wfi_,rfi_,reh,rel,xrdy,xreor].clk = clock0;

        [xci0,xci1,xci2,lst,lst_,xrd_,strb,q0].clk   = clock0;

state_diagram sregA;

"idle loop
"=========
      state A01:
        IF (!bchsl) THEN A01;                    "wait for chsl
        IF (bchsl & !bchrd) THEN A10;            "chsl,write cycle
        IF (bchsl & bchrd) THEN A20;             "chsl,read cycle

"write cycle
"===========
      state A10:"xrdy
        IF (bchsl & !bwdat) THEN A10;             "wait for wdat
        IF (bchsl & bwdat & !bsw16) THEN A13;     "byte write cycle
        IF (bchsl & bwdat &  bsw16) THEN A11;     "word write cycle
        IF (!bchsl) THEN A01;                     "!chsl,forced return

      state A11:"weh_
        IF (bchsl & !bffr) THEN A11;              "wait for ffr
        IF (bchsl & bffr) THEN A12;
        IF (!bchsl) THEN A01;                     "!chsl,forced return

      state A12:"weh_,wfi_
        GOTO A13;

      state A13:"wel_
        IF (bchsl & !bffr) THEN A13;              "wait for ffr
        IF (bchsl & bffr) THEN A14;
        IF (!bchsl) THEN A01;                     "!chsl,forced return

      state A14:"wel_,wfi_
        GOTO A10;

"read cycle
"==========

      state A20:
        IF (bchsl & !bfda) THEN A20;             "wait for fda
        IF (bchsl & bfda & brdt_) THEN A20;      "wait for !rdt_
        IF (bchsl & bfda & !brdt_) THEN A21;
        IF (!bchsl) THEN A01;                    "!chsl,forced return

      state A21:"reh_,rfi_
        IF (!bsw16 & !reor) THEN A24;           "byte read cycle
```

```
      IF (bsw16 & !reor) THEN A22;          "word read cycle
      IF (reor) THEN A30;                    "end of record

   state A22:
      IF (bchsl & !bfda) THEN A22;           "wait for fda
      IF (bchsl & bfda) THEN A23;
      IF (!bchsl) THEN A01;                  "!chsl,forced return

    state A23:"rel_,rfi_
      IF (!reor) THEN A24;
      IF (reor) THEN A30;                    "end of record

    state A24:"xrdy
      IF (bchsl & !brdt_) THEN A24;          "wait for rdt_
      IF (bchsl & brdt_) THEN A20;
      IF (!chsl) THEN A01;                "!chsl,forced return

    state A30:"xreor
      IF (bchsl) THEN A30;                  "wait for !chsl
      IF (!bchsl) THEN A01;                 "!chsl,forced return
state_diagram sregB;
"wait for request
"================

    state B01:
      GOTO B02;                             "delay cycle

    state B02:"request handling
  IF (back) THEN B02;
  IF (!brqrr & !brqsta & !brqdt & !brqstp & !brqgo) THEN B02;
  IF (brqrr & !back) THEN B10;
  IF (!brqrr & brqsta & !back) THEN B20;
  IF (!brqrr & !brqsta & brqstp & !back) THEN B30;
  IF (!brqrr & !brqsta & !brqstp & brqgo & !back) THEN B40;
  IF (!brqrr & !brqsta & brqdt & !brqstp & !brqgo & bstopx &!back) THEN B02;
  IF (!brqrr & !brqsta & brqdt & !brqstp & !brqgo & !bstopx &!back) THEN B50;
" 'init' request
"===============

    state B10:"xci0,xci1
      GOTO B11;

    state B11:"xci0,xci1,strb
      GOTO B01;
" 'status' request
"=================

    state B20:"xci2,lst,lst_
      GOTO B21;

    state B21:"xci2,lst_,strb
      GOTO B01;
" 'xoff' request
"===============

    state B30:"xci0
      GOTO B31;

    state B31:"xci0,strb
      GOTO B01;
" 'xon' request
"==============

    state B40:"xci1
      GOTO B41;

    state B41:"xci1,strb
      GOTO B01;
```

```
" 'data' request
"===============

     state B50:"xrd_
       GOTO B51;

     state B51:"xrd_,strb
       GOTO B02;

test_vectors "read/write - byte/word control
"([clock0,sw16,chsl,chrd,ffr,wdat,fda,rdt_,reor]->[sregA])
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
" [ .c. , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
" [ .c. , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 0 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 0 , 1 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ A13 ];
" [ .c. , 0 , 1 , 0 , 1 , 1 , 0 , 0 , 0 ]->[ A13 ];
" [ .c. , 0 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A14 ];
" [ .c. , 0 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 0 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 0 , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
"------------------------------------------------------------
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
" [ .c. , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
" [ .c. , 1 , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 1 , 1 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 1 , 1 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ A11 ];
" [ .c. , 1 , 1 , 0 , 1 , 1 , 0 , 0 , 0 ]->[ A11 ];
" [ .c. , 1 , 1 , 0 , 1 , 1 , 0 , 0 , 0 ]->[ A12 ];
" [ .c. , 1 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A13 ];
" [ .c. , 1 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A14 ];
" [ .c. , 1 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 1 , 1 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A10 ];
" [ .c. , 1 , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ A01 ];
"------------------------------------------------------------
" clock0,sw16,chsl,chrd,ffr,wdat,fda,rdt_,reor
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 0 , 1 , 0 ]->[ A01 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 0 , 1 , 0 ]->[ A20 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A20 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A20 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A20 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A21 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A24 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A24 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A24 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A20 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A20 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 1 ]->[ A21 ];
" [ .c. , 0 , 1 , 1 , 0 , 0 , 1 , 0 , 1 ]->[ A30 ];
" [ .c. , 0 , 0 , 1 , 0 , 0 , 1 , 0 , 1 ]->[ A30 ];
" [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
"------------------------------------------------------------
" [ .c. , 0 , 0 , 0 , 0 , 0 , 1 , 0 , 0 ]->[ A01 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A01 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A20 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A20 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A20 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A21 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A22 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A23 ];
```

```
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A24 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A24 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 1 , 0 ]->[ A20 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A20 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A21 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A22 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 0 ]->[ A23 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0   1 ]->[ A30 ];
" [ .c. , 1 , 1 , 1 , 0 , 0 , 1 , 0 , 1 ]->[ A30 ];
" [ .c. , 1 , 0 , 1 , 0 , 0 , 1 , 0 , 1 ]->[ A30 ];
" [ .c. , 1 , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ A01 ];
"=============================================================

"test_vectors  request control

([clock0,rqrr,rqsta,rqdt,rqstp,rqgo,stopx,ack]->[sregB])
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 1 ]->[ B02 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B10 ];
 [ .c. , 1 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B11 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B01 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 1 , 1 , 1 , 1 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 1 , 1 , 1 , 1 , 0 , 0 ]->[ B20 ];
 [ .c. , 0 , 1 , 1 , 1 , 1 , 0 , 0 ]->[ B21 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B01 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ B30 ];
 [ .c. , 0 , 0 , 0 , 1 , 0 , 0 , 0 ]->[ B31 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B01 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 0 , 0 , 1 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 0 , 0 , 1 , 0 , 0 ]->[ B40 ];
 [ .c. , 0 , 0 , 0 , 0 , 1 , 0 , 0 ]->[ B41 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B01 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ B02 ];
 [ .c. , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ B50 ];
 [ .c. , 0 , 0 , 1 , 0 , 0 , 0 , 0 ]->[ B51 ];
 [ .c. , 0 , 0 , 0 , 0 , 0 , 0 , 0 ]->[ B02 ];

end
```

FITTER FILE: MACH210

Note: This file has been condensed in order to save trees.

```
AMD MACH FITR - MARKET RELEASE (1-24-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990
Flags Used:            Unplace=False          Max Packing=True
Flags Used:         Expand Small=False        Expand All=False
**************************************************************************************
     Mach PLD Fitter - v 1.46          HSSL2, Transmission and byte/word control
**************************************************************************************
*** Timing Analysis for Signals
 Parameter    Min    Max          Signal List (Those having Max delay.)
     Tsu       1      1          bsw16          bchsl          bchrd
                                  bffr           bwdat          bfda
                                 brdt_           back
     Tco       0      0           wel_           weh_           wfi_
                                  rfi_           reh            rel
                                  xrdy           q0
     Tcr       1      1           wel_           weh_           xrdy
                                  wfi_           rfi_           rel
                                   reh           strb
 .
 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Register clock to combinatorial output
 Tcr - Register thru combinatorial logic to setup
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                Available          Used          Remaining
        Clocks:     2                1               1
          Pins:    38               32               6       ->     84%
     I/O Macro:    32               16              16
   Total Macro:    64               30              34
 Product Terms:   256               66             124       ->     50%

MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
*** Last Equations Placed in Blocks
Weakly -
Assign -          bsw16           bchsl           bchrd           bffr
Assign -          bwdat           bfda            brdt_           brqrr
*** Block Partitioning Results
                Array    Macros    # I/O    Buried   Product    Signal
               Inputs    Remain    Macro    Logic     Terms     Fanout
   Block-> A     16        8         8        0        44         8
   Block-> B     15        8         8        0        32         8
   Block-> C      7        9         0        7        28         7
   Block-> D      7        9         0        7        28         7
*** Block Signal List
Block-> A         xreor            rel            wfi_            reh
                   rfi_            weh_           xrdy            wel_
Block-> B            q0            strb           lst_            lst
                   xci2            xci0           xrd_            xci1
```

```
Block-> C          bstopx           brqstp           brqsta          brdt_
                    bwdat            bchsl            bsw16

Block-> D            back            brqgo            brqdt           brqrr
                     bfda             bffr            bchrd

|> INFORMATION F050 - Device Utilization....... *: 62 %

*** Feedback Map - HSSL2, Transmission and byte/word control

 Gbl Inp .--.       I/O   .--+--A--+--.   I/O             I/O  .--+--B--+--.  I/O
         | 0|       rel : 0|       |21| bffr            xci1 : 0|       |21| brqsta
         | 1|      weh_ : 1|       |20|                      : 1|       |20|
         | 2|      wfi_ : 2|       |19|                   q0 : 2|       |19|
         | 3|      xrdy : 3|       |18|                 strb : 3|       |18|
         | 4|       reh : 4|       |17|               bstopx : 4|       |17|
         | 5|      wel_ : 5|       |16|                 xci2 : 5|       |16|
         '--'      rfi_ : 6|       |15: bfda            back : 6|       |15|
                  xreor : 7|       |14: bchrd           xci0 : 7|       |14: brqstp
                        | 8|       |13: bsw16           lst_ : 8|       |13: brqrr
                  brdt_ : 9|       |12: bchsl           xrd_ : 9|       |12: brqdt
                   reor :10|       |11: bwdat            lst :10|       |11: brqgo
                          '--+--C--+--'                       '--+--D--+--'
                  stopx : 0|       |21|                  fda : 0|       |21|
                  rqsta : 1|       |20|                      | 1|       |20|
                        | 2|       |19|                  ack : 2|       |19|
                   wdat : 3|       |18|                 chrd : 3|       |18|
                   sw16 : 4|       |17|                 rqrr : 4|       |17|
                  rqstp : 5|       |16|                  ffr : 5|       |16|
                        | 6|       |15|                 rqdt : 6|       |15|
                   rdt_ : 7|       |14|                      | 7|       |14: rqgo
                   chsl : 8|       |13|                      | 8|       |13|
                        | 9|       |12|                      | 9|       |12|
                        |10|       |11|                      |10|       |11|
                          '--+-u--u+--'                       '--+-u--u+--'
```

```
*** Logic Map - HSSL2, Transmission and byte/word control

 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
     reor| 0|      rel | 0| 7   |21|                  q0 | 0| 3   |21|
    stopx| 1|          | 1| *   |20|                     | 1| .   |20|
   clock0| 2|     xrdy | 2| 6   |19|                strb | 2| 4   |19|
     chrd| 3|          | 3| *   |18|                     | 3|     |18|
     rqrr| 4|     wel_ | 4| 7   |17|                lst_ | 4| 2   |17|
      ffr| 5|          | 5| *   |16|                     | 5| .   |16|
         '--'    xreor | 6| 3  .|15|                 lst | 6| 1  .|15|
                       | 7| .  4|14| weh_                | 7| .  3|14| xci1
                  wfi_ | 8| 2  .|13|                xci2 | 8| 2  .|13|
                       | 9| .  2|12| rfi_                | 9| .  2|12| xrd_
                   reh |10| 1  .|11|                xci0 |10| 3  .|11|
                       '--+--C--+--'                     '--+--D--+--'
                brqstp | 0| 1   |21|                     | 0| .   |21|
                       | 1| .   |20|                bffr | 1| 1   |20|
                 brdt_ | 2| 1   |19|               brqgo | 2| 1   |19|
                       | 3| .   |18|               brqdt | 3| 1   |18|
                 bchsl | 4| 1   |17|               brqrr | 4| 1   |17|
                       | 5| .   |16|               bchrd | 5| 1   |16|
                 bsw16 | 6| 1  1|15| bstopx         bfda | 6| 1  .|15|
                       | 7| .  1|14| bwdat               | 7| .  1|14| back
                       | 8| .  1|13| brqsta              | 8| .  .|13|
                       | 9| .  .|12|                     | 9| .  .|12|
                       |10| .  .|11|                     |10| .  .|11|
                       '--+-u--u+--'                     '--+-u--u+--'
```

```
*** Pin Map - HSSL2, Transmission and byte/word control

                      rel
                  xrdy  |        rdt_
               wel_  |  |        |  chsl
           xreor  |  |  |        |  |  fda
         wfi_  |  |  |  |        |  |  |  rqsta
            |  |  |  |  |        |  |  |  |
      .-----'--'--'--'--'--o-----'--'--'--'---.
      |                       4  4  4  4  4   |
      |     6  5  4  3  2  1  4  3  2  1  0   |
   reh| 7                                   39|ack
  rfi_| 8                  G  V             38|wdat
  weh_| 9                  n  c             37|sw16
  reor|10                  d  c             36|rqgo
 stopx|11                                   35|ffr
  Gnd |12            MACH-210               34| Gnd
clock0|13                                   33|rqrr
  xci1|14             V  G                  32|chrd
  xrd_|15             c  n                  31|
  xci0|16             c  d                  30|
  xci2|17                                   29|
      |   1  1  2  2  2  2  2  2  2  2  2     |
      |   8  9  0  1  2  3  4  5  6  7  8     |
      '---.--.--.--.--.-----.--.--.--.--.-----'
          |  |  |  |        |  |  |  |  |
        lst  |  |  |        |  |  |  |  '
          lst_  |  |        |  |  |  '
             strb  |        |  |  '
                  q0        |  rqstp
                            rqdt
```

```
The Design Doc is  stored in ===> hssl2.Rpt
The Jedec Data is  stored in ===> hssl2.Jed
The Placements are stored in ===> hssl2.Plc
The  Fuse Plot is  stored in ===> hssl2.Xpt

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: hssl2
```

PLACEMENT FILE: MACH210

```
; Flags Used:          Unplace=False        Max Packing=True
; Flags Used:      Expand Small=False        Expand All=False

; FITR generated placements
Pin    13          clock0  ; Inp          ; I  2
Pin    37            sw16  ; Inp          ; D 12
Pin    42            chsl  ; Inp          ; D  2
Pin    32            chrd  ; Inp          ; I  3
Pin    35             ffr  ; Inp          ; I  5
Pin    38            wdat  ; Inp          ; D 10
Pin    41             fda  ; Inp          ; D  4
Pin    43            rdt_  ; Inp          ; D  0
Pin    10            reor  ; Inp          ; I  0
Pin    33            rqrr  ; Inp          ; I  4
Pin    40           rqsta  ; Inp          ; D  6
Pin    24            rqdt  ; Inp          ; C  0
Pin    25           rqstp  ; Inp          ; C  2
Pin    36            rqgo  ; Inp          ; D 14
Pin    11           stopx  ; Inp          ; I  1
Pin    39             ack  ; Inp          ; D  8
Node    6          wel__I    Reg          ; A  4
Node   16          weh__I    Reg          ; A 14
Node   10          wfi__I    Reg          ; A  8
Node   14          rfi__I    Reg          ; A 12
Node    2           rel_I    Reg          ; A  0
Node   12           reh_I    Reg          ; A 10
Node    4          xrdy_I    Reg          ; A  2
Node    8         xreor_I    Reg          ; A  6
Node   28          xci0_I    Reg          ; B 10
Node   32          xci1_I    Reg          ; B 14
Node   26          xci2_I    Reg          ; B  8
Node   24           lst_I    Reg          ; B  6
Node   22          lst__I    Reg          ; B  4
Node   30          xrd__I    Reg          ; B 12
Node   20          strb_I    Reg          ; B  2
Node   18            q0_I    Reg          ; B  0
Node   40           bsw16    Reg          ; C  6
Node   38           bchsl    Reg          ; C  4
Node   55           bchrd    Reg          ; D  5
Node   51            bffr    Reg          ; D  1
Node   48           bwdat    Reg          ; C 14
Node   56            bfda    Reg          ; D  6
Node   36           brdt_    Reg          ; C  2
Node   54           brqrr    Reg          ; D  4
Node   47          brqsta    Reg          ; C 13
Node   53           brqdt    Reg          ; D  3
Node   34          brqstp    Reg          ; C  0
Node   52           brqgo    Reg          ; D  2
Node   49          bstopx    Reg          ; C 15
Node   64            back    Reg          ; D 14
Pin     4            wel_    Reg          ; A  4
Pin     9            weh_    Reg          ; A 14
Pin     6            wfi_    Reg          ; A  8
Pin     8            rfi_    Reg          ; A 12
Pin     7             reh    Reg          ; A 10
Pin     2             rel    Reg          ; A  0
Pin     3            xrdy    Reg          ; A  2
Pin     5           xreor    Reg          ; A  6
Pin    16            xci0    Reg          ; B 10
Pin    14            xci1    Reg          ; B 14
Pin    17            xci2    Reg          ; B  8
```

```
Pin   18          lst    Reg        ; B  6
Pin   19         lst_    Reg        ; B  4
Pin   15         xrd_    Reg        ; B 12
Pin   20         strb    Reg        ; B  2
Pin   21           q0    Reg        ; B  0

; Group Mach_Seg_A xreor rel wfi_ reh rfi_ weh_ xrdy wel_
; Group Mach_Seg_B q0 strb lst_ lst xci2 xci0 xrd_ xci1
; Group Mach_Seg_C bstopx brqstp brqsta brdt_ bwdat bchsl bsw16
; Group Mach_Seg_D back brqgo brqdt brqrr bfda bffr bchrd
```

Self-Timing ISA-Bus Interface: MACH110 and MACH210

Advanced Micro Devices

Application Note

by Ulrich Lonsdorfer, Germany

PURPOSE OF THE INTERFACE

One method for enhancing the performance of computers and of expanding their applications is the use of special, auxiliary processors. In order to make an expedient use of these devices, a high-performance communications channel is required. One proven device is memory-accessible from two sides. To implement this solution, special dual-port-RAM components are only suitable for smaller memory areas. In case of larger memory requirements, we usually rely on conventional memories. The dual-port function is constructed from bus drivers with appropriate controller. The most important function of this type of controller is the arbitration between the two masters. In case of simultaneous access to the common memory, one side has to wait. Now to prevent this where possible, bus drivers with registers are being used in the described interface. These registers are used as one-stage caches and accelerate the transfers without additional expense.

The individual processes will be explained using the example of a write-cycle. If the address decoder reports a write access to the common memory area, then data and addresses will be stored in the drivers and the external cycle will be completed with no waiting cycles. Then a wait is required until the internal bus is free. Once this is the case, the drivers will be released and the datum will be placed in RAM through operation of the write-line. If a new access occurs within this time, waiting cycles are unavoidable.

In case of optional read cycles, this method can not be applied, because a prediction would be needed. In case of sequential reads, addresses will be held internally and incremented after every access. The datum will be reserved in the driver and can be read without waiting.

As a practical example, we selected a PC add-on board. This board makes available a dual ported memory of 1 Mbit in the extended memory that allows 16-bit access. The controller for this board will be implemented by using the MACH110 component. The fundamental circuit is shown in Figure 6-1.

FUNCTION OF THE INTERFACE CONTROLLER

To control this interface the following functions should be implemented in the MACH component:

- Address decoder for memory area
- Address decoder for I/O area
- Acknowledge for 16-bit access
- Control of the waiting cycles
- Control of the bus driver
- Generation of write and read access

The address decoder selects a 1 Mbit sized memory area from the 16 Mbit large address area of the ISA bus, by linking the unlatched addresses A19 – A23 with AEN. AEN indicates refresh cycles on the bus that are not needed for this board. Linked with the write and read lines, the signals MEM_RD_ZY and MEM_WR_ZY are produced that are allocated to one node, since they are needed only internally. The same applies for IO_RD_ZY and IO_WR_ZY that are formed from the latched addresses A0 – A9, AEN, IOW~ and IOR~. The Ack signals MEMCS16~ and IOCS16~ tell the motherboard that the card can perform 16 bit transfers. These are particularly time-critical and are thus designed as a single-step and are not linked with the actual address decoder. Since they may only be active when the card is selected, their output driver will be controlled by the signal itself, and thus an open collector simulation is achieved.

Control of the bus driver, of the waiting cycles and of the internal memory access are closely linked together. Therefore, the first implementation test was in a state machine with all starting quantities as state vectors. This attempt failed merely due to the available P-terms.

Today's solution with two linked, much simpler state machines is more elegant. The one for the external bus controls enable and latch signals for the bus driver and the zero wait and wait signals. The one for the internal bus controls the internal enable and latch signals and also the internal write and read line. This solution offers the additional advantage of driving each machine with an independent clock, since the MACH components have several clock inputs.

Publication# 17755 Rev. A Amendment/0
Issue Date: May 1993

STATES OF THE INTERFACE CONTROLLER

The diagrams of state for the two machines are shown in Figures 6-2 and 6-3. These figures were produced with a special program that converts the generated graphic immediately into the attached text-form in ABEL syntax. My thanks to my collegue Mr. Gerhard Lehnerts for making this program available.

From the symmetry of the diagram for the external machine, we see that we are dealing with four very similar transitions. For example, let us look at the register read. From the reset and idle state, the machine will start when the address decoder reports an access; the machine persists in the next state until it is completed. Then it goes into the next state and waits until the internal machine signals READY. In this state, the external bus must be stopped with Wait, since the interface is not free.

The machine for the internal bus is much simpler. It starts from Reset and Idle state when the external machine is in the Wait state and at the same time, the internal bus is reported free. Then come two-step write and read access without additional conditions.

The vectors of state of the two machines are composed of the functional outputs. Auxiliary variables are not required before READY. These variables of the internal machine tell the external one that the internal process is completed, and that it can go over into the free state.

Signaling in the reverse direction takes place with RD_REQUEST and WR_REQUEST. As described in the "Conventions" section, these are formed from the quantities of state WAIT~ and DEX_LAT of the one machine without the use of macrocells. Since the pins of the two outputs are controlled with OE, naturally the Feedback signal must be used.

IMPLEMENTATION

The complete design fits with no additional effort into the component MACH210 and fulfills at least the formulated test vectors.

Since the degree of utilization in all resources rests at less than 50%, an attempt should be made also to bring the design into a MACH110. Without the controller, none of the four strategies of the fitter can meet this goal. The essential bottleneck here is the bottleneck into the two blocks of the component.

In order to select the best strategy, I first reduced the scope of the design in order to achieve a fit. This was particularly easy given the level of the address decoder. The best strategy turned out to be the "full expand." In addition, I distributed the macrocells by means of the Group command, to the blocks in order to reduce the number of inputs. To do this, I placed every state machine into a separate block and filled the remaining space with the address decoders. Thus, I was able to bring back in additional functions of the original design.

Although according to the Report File several inputs are still free, I had to leave off several details. Every additional address in the IO decoder causes a No Fit. The output control for MEMCS16~ and IOCS16~ also had to be simplified so that no return is required here.

SUMMARY

According to experience, the existence of the test vectors is no guarantee for proper functioning in the real circuit. Therefore I would like to build and test the design. Not least therefore, I have selected the PC bus as the object of the test.

Beschreibung des Self-Timing ISA-Interfaces: MACH110 and MACH210

Advanced Micro Devices

Application Note

by Ulrich Lonsdorfer, Germany

ZWECK DES INTERFACES

Ein Mittel um die Leistungsfähigkeit von Rechnern zu steigern und ihren Einsatzbereich zu erweitern, sind spezielle Zusatzprozessoren. Um diese sinnvoll einsetzen zu können ist ein leistungsfähiger Kommunikatinskanal notwendig. Eine bewährtes Mittel ist ein von zwei Seiten zugreifbarer Speicher. Zur Realisierung sind spezielle Dual-Port-Ram Bausteine nur für kleinere Speicherbereiche geeignet. Bei größerem Speicherbedarf greift man auf konventionellen Speicher zurück. Die Dual-Port-Funktion baut man durch Bus-Treiber mit entsprechender Ansteuerung auf. Die wichtigste Funktion eines solchen Controlers ist das Arbitrieren zwischen den beiden Mastern. Bei gleichzeitigem Zugriff auf den gemeinsamen Speicherbereich muß eine Seite warten. Um das nach Möglichkeit zu vermeiden, werden in dem beschriebenen Interface Bustreiber mit Registern eingesetzt. Diese dienen als einstufige Caches und beschleunigen die Transfers ohne zusätzlichen Aufwand.

Die enzelnen Vorgänge sollen am Beispiel eines Schreibzyklus erläutert werden. Meldet der Adressdekoder einen Schreibzugriff auf den gemeinsamen Speicherbereich, werden Daten und Adressen in den Treibern gespeichert, und der externe Zyklus wird ohne Wartezyklen abgeschlossen. Dann muß gewartet werden, bis der interne Bus frei ist. Wenn das der Fall ist, werden die Treiber freigegeben und durch bedienen der Schreibleitung das Datum im RAM abgelet. Erfolgt innerhalb dieser Zeit ein erneuter Zugriff sind Wartezyklen unvermeidlich.

Bei wahlfreinen Lesezyklen kann diese Verfahren nicht angewendet werden, weil eine Prädiktion notwendig wäre. Bei sequentiellem Lesen werden Adressen intern gahalten und bei jedem Zugriff inkrementiert. Das Datum wird im Treiber vorgehalten und kann ohne Warten gelesen werden.

Als Realisierungsbeispiel wurde ein PC-Zusatzboard gewählt. Dieses stellt einen Dual-Ported-Speicher von 1 MBit im Extended Memory zur Verfügung, auf den 16-Bit breit zugegriffen werden kann. Der Controller dafür soll mit Hilfe des MACH110 Bausteins realisiert werden. Die prinzipielle Schaltung zeigt Figure 6-1.

FUNKTION DES INTERFACE CONTROLLERS

Zur Steuerung dieses Interfaces sollen im MACH-Baustein folgende Funktionen realisiert werden:

- Adressdekoder für Memory-Bereich
- Adressdekoder für I/O-Bereich
- Acknowledge für 16 Bit Zugriffe
- Steuerung der Wartezyklen
- Steuerung der Bustreiber
- Generieren von Schreib- und Lesezugriffen

Der Adressdekoder wählt einen 1 M-Bit großen Memory-Bereich aus dem 16 Mbit großen Adreßraum des ISA-Busses, indem er die ungelatchten Adressen A19 – A23 mit AEN verknüpft. AEN zeigt Refresh-Zyklen auf dem Bus an, die für dieses Board nicht benötigt werden. Mit den Schreib- und Leseleitungen verknüpft entstehen die Signale MEM_RD_ZY und MEM_WR_ZY, die einem Node zugeordnet sind, da sie nur intern gebraucht werden. Das gleiche gilt für IO_RD_ZY und IO_WR_ZY, die aus den gelatchten Adressen A0 – A9, AEN, IOW und IOR gebildet werden. Die Acknowlesignale MEMCS16 und IOCS16 zeigen dem Motherboard an, daß die Karte 16-Bit Transfers durchführen kann. Diese sind besonders zeitkritisch und sind deshalb einstufig ausgeführt und nicht mit dem eigentlichen Adreßdekoder verknüpft. Da sie nur aktiv werden dürfen, wenn die Karte selektiert ist, wird ihr Ausgangstreiber vom Signal selber gesteuert, und so eine Open Kollektor Simulation erzielt.

Die Steuerung der Bustreiber, der Wartezyklen und die internen Memoryzugriffe sind eng miteinander verknüpft. Deshalb war der erste Realisierungsversuch eine Zustandsmaschine mit allen Ausgangsgrößen als Zustandsvektor. Dieser scheiterte allein schon an den zur Verfügung stehenden P-Termen. Handlicher ist die jetzige Lösung mit zwei verkoppelten, wesentlich simpleren Zustandsmaschinen. Die für den externen Bus steuert Enable- und Latchsignale für die Bustreiber und die Zero-Wait und Wait Signale. Die für den internen Bus steuert die internen Enable- und Latchsignale sowie die interne Schreib- und Leseleitung. Diese Lösung bietet den zusätzlichen Vorteil, jede Maschine mit einem unabhängigen Takt zu betreiben, da die MACH Bausteine mehrere Takteingänge haben.

ZUSTÄNDE DES INTERFACE-CONTROLERS

Die Zustandsdiagramme für die beiden Maschinen zeigen die Figure 6-2 und Figure 6-3. Diese wurden mit einem speziellen Programm erzeugt, das die erzeugte Grafik sofort in die beigefügte Textform in ABEL-Syntax umsetzt. Für die Überlassung diese Programms danke ich meinem Kollegen Herrn Dipl. Ing. Gerhard Lehnerts.

Aus der Symmetrie des Diagramms für die externe Maschine erkennt man, daß es sich um 4 sehr ähnliche Übergänge handelt. Beispielhaft soll das Registerlesen beschrieben werden. Aus dem Reset und Idle Zustand startet die Maschine, wenn die Adressdekoder einen Zugriff melden. im nächsten Zustand verharrt die Maschine, bis er beendet ist. Dann geht sie in den nächsten Zustand und wartet bis die interne Maschine READY signaliert. In diesem Zustand muß der externe Bus mit Wait angehalten werden, da das Interface nicht frei ist.

Die Maschine für den internen Bus ist noch wesentlich einfacher. Sie startet aus Reset- und Idlezustand, wenn die externe Maschine im Wartezustand ist und gleichzeitig der interne Bus als frei gemeldet wird. Dann folgen zweistufige Schreib- Lesezugriffe ohne weitere Bedingungen.

Die Zustandsvektoren beider Maschinen sind aus den funktionellen Ausgängen zusammengesetzt. Hilfsvariable sind bis auf READY nicht notwendig. Diese Variable der internen Maschine zeigt der externen an, daß der interne Vorgang abgeschlossen ist, und daß sie in den Frei-Zustand übergehen kann.

Die Signalisierung in die umgekehrte Richtung erfolgt mit RD_REQUEST und WR_REQUEST. Diese werden, wie im Vereinbarungsteil beschrieben, aus den Zustandsgrössen WAIT und DEX_LAT der externen Maschine gebildet ohne Macrozellen zu verbrauchen.

Da die Pins der beiden Ausgänge mit OE gesteuert werden, muß natürlich das Feedback-Signal verwendet werden.

IMPLEMENTIERUNG

Das komplette Design passt ohne weiteres zutun in den Baustein MACH210 und erfüllt die formulierten Testvektoren.

Da der Ausnutzungsgrad in allen Resourcen bei weniger als 50 % liegt, sollte versucht werden, den Entwurf auch in einen MACH110 zu bringen. Ohne Steuerung kann keine der 4 Strategien des Fitters diese Ziel erreichen. Der wesentliche Engpass sind dabei die Eingänge in die beiden Blocks des Bausteins. Um die beste Strategie auszuwählen habe ich zunächst den Umfang des Designs reduziert, um überhaupt ein Fitten zu erzielen. Das war bei der Tiefe des Adressdekoders besonders bequem. Als beste Strategie stellt sich "Full expand" heraus. Weiterhin habe ich die Macrozellen mit Hilfe des Group-Befehls auf die Blocks verteilt, um Eingänge zu sparen. Dazu habe ich jede Zustandsmaschine in einen eigenen Block plaziert und den verbleibenden Raum mit den Adressdekodern gefüllt. Dadurch konnte ich weitere Funktionen des ursprünglichen Designs wieder hinzunehmen. Obwohl laut Report-File noch Eingänge frei sind, mußte ich auf einige Details verzichten. Jede weitere Adresse im IO-Dekoder bewirkt ein Nofit. Auch die Ausganssteuerung für MEMCS16 und IOCS16 mußte vereinfacht werden, damit hier keine Rückführung notwendig ist.

RESUME

Erfahrungsgemäß ist das bestehen der Testvektoren noch keine Garantie für die Funktion in der realen Schaltung. Deshalb möchte ich den Entwurf auch aufbauen und testen. Nicht zuletzt deshalb habe ich den PC-Bus als Versuchsobjekt gewählt.

STATE DIAGRAM STATE REGISTER

```
STATE EXTERN-FREE:                    "Is Reset Entry
if (10 RD CYCLE)                      thenREAD-REGlSTER
else if (MEM RD CYCLE)                then READ-MEMORY
else if (10 WR CYCLE)                 then WRITE-REGISTER
else if (MEM WR CYCLE)                then WRITE-MEMORY
else EXTERN-FREE;                     "Default State

STATE READ-REGISTER:
i f (110 RD CYCLE)                    then WAIT-READ-REGISTER
else READ-REGISTER;                   "Remaining possibilities

STATE WAIT-READ-REGISTER:
i f (IREADY) then WAIT-READ-REGISTER
else EXTERN-FREE;                     "Remaining possibilities

STATE WRITE-REGISTER:
if (110 WR CYCLE)                     then WAIT-WRITE-REGISTER
else WRITE-REGISTER;                  "Remaining possibilities

STATE WAIT-WRITE-REGISTER:
If (IREADY) then WAIT-WRITE-REGISTER
else EXTERN-FREE;                     "Remaining possibilities

STATE READMEMORY:
i f (IMEM RD CYCLE)                   then WAIT-READ-MEMORY
else READ-MEMORY;                     "Remaining possibilities

STATE WAIT-READ-MEMORY:
if (IREADY)                           then WAIT-READ-MEMORY
else EXTERN-FREE;                     "Remaining possibilities

STATE WRITE-MEMORY:
i f (IMEM WR CYCLE)                   then WAIT-WRITE-MEMORY
else WRITE-MEMORY;                    "Remaining possibilities

STATE WAIT-WRITE-MEMORY:
if (IREADY) then WAIT-WRITE-MEMORY
else EXTERN-FREE;                     "Remaining possibilities

STATE INTERN-IDLE:                    "is Reset Entry
if (WR REQUEST & IFREE)               then INTERN-WRITE1
else if (RD REQUEST & IFREE)          then INTERN-READ1
else INTERN-lDLE;                     "Default State

STATE INTERN-READ1:
goto INTERN-READ2;                    "Pass always

STATE INTERN-READ2:
goto INTERN-lDLE;                     "Pass always

STATE INTERN-WRITE1:
goto INTERN-WRITE2;                   "Pass always

STATE INTERN-WRITE2:                  go to INTERN-lDLE; "Pass always
```

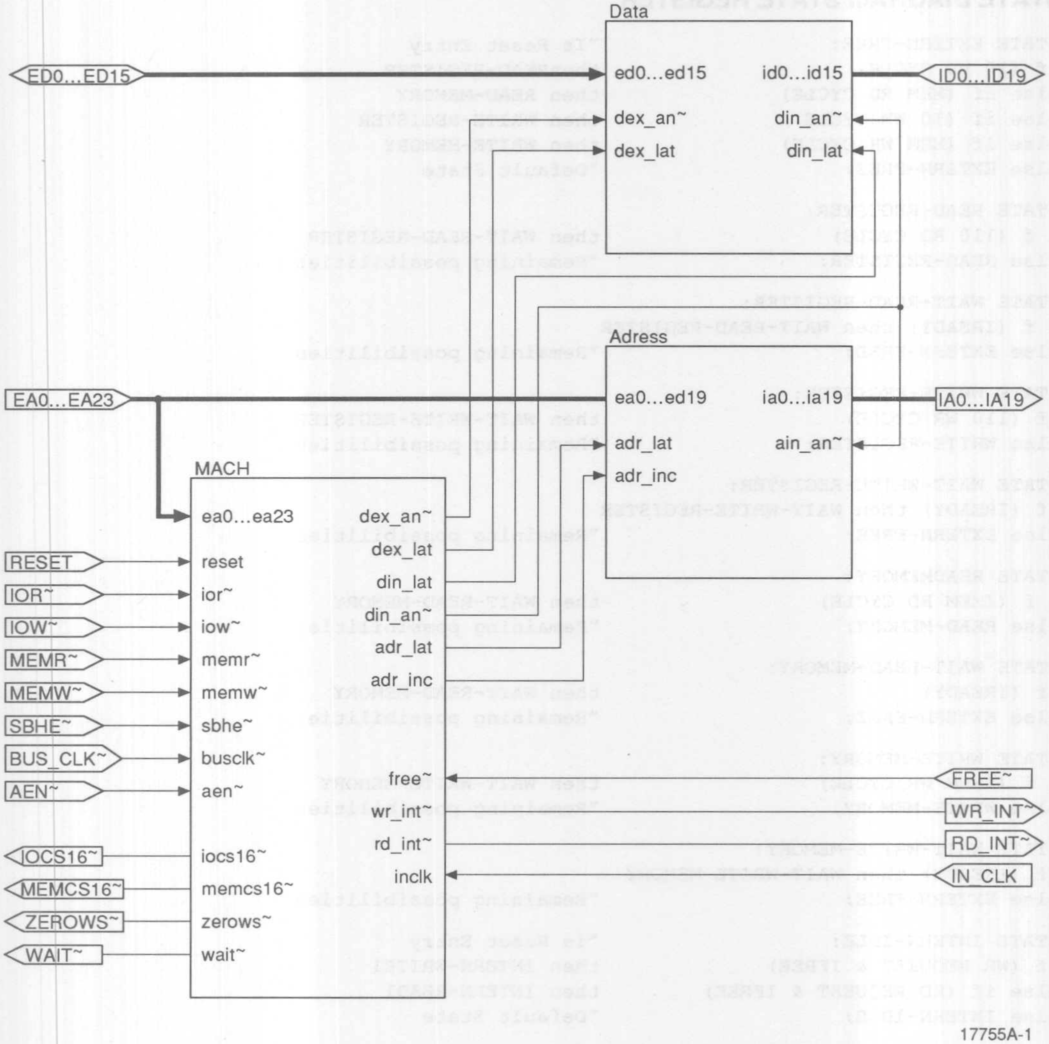


Figure 6-1. Fundamental Circuit of the ISA Interface

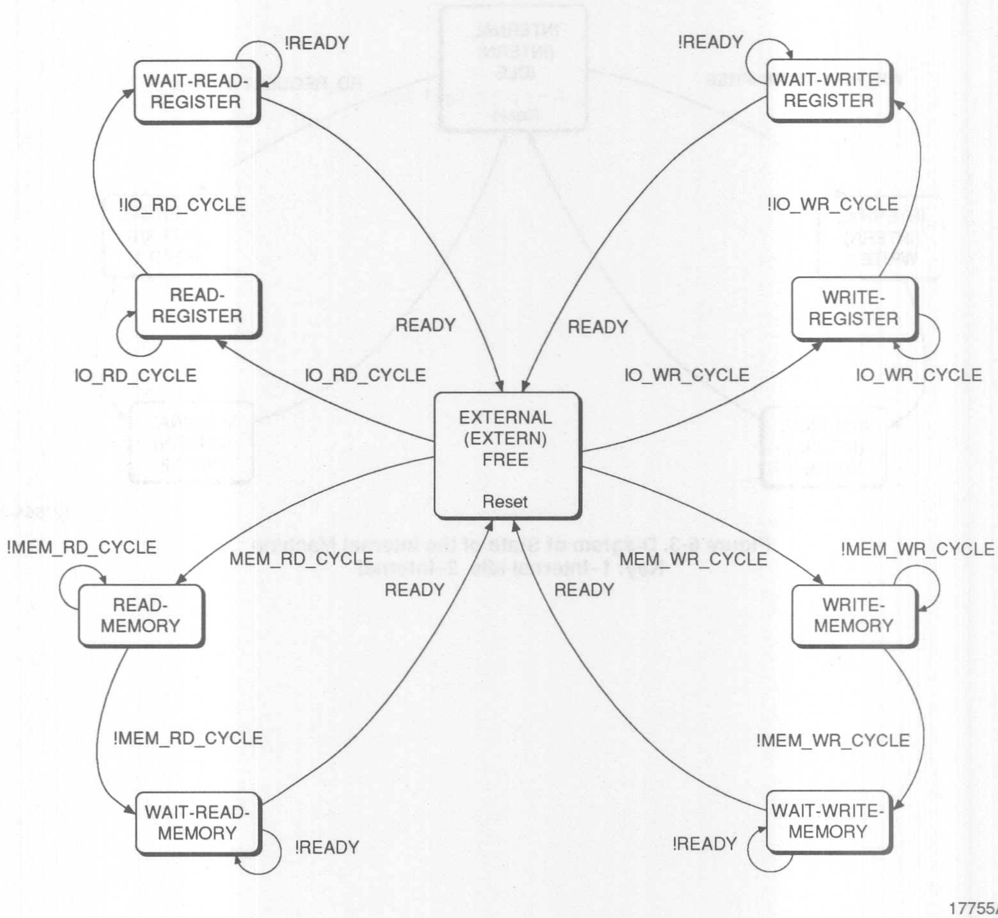


Figure 6-2. Diagram of State of the External Machine
Key: 1–External Free

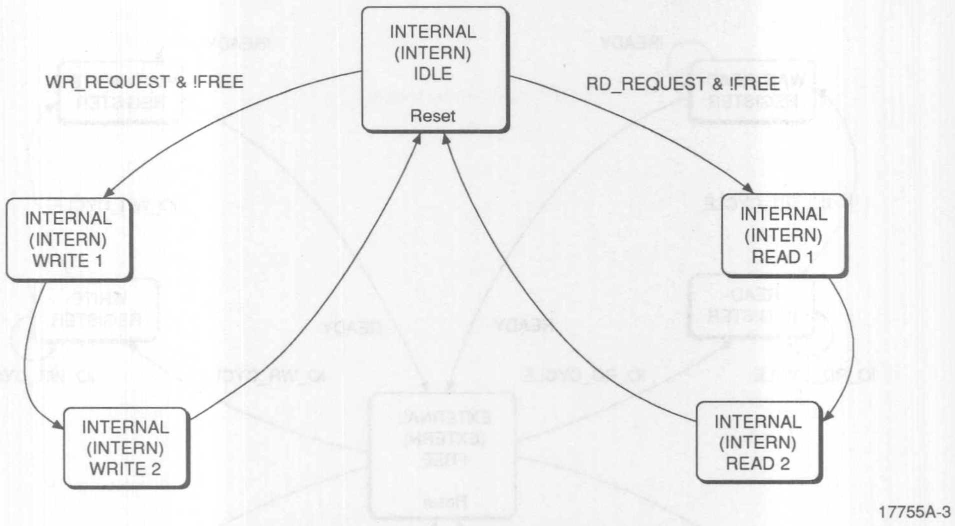


Figure 6-3. Diagram of State of the Internal Machine
Key: 1–Internal Idle 2–Internal

DESIGN FILE: MACH110

```
module ISA
title  'ISA-BUS-INTERFACE'
declarations
"-----------------------------------------------------------------------

ISA      device 'mach110';

"Eing nge
"-----------------------------------------------------------------------
EXTAKT   pin           ;"Bustakt
INTAKT   pin           ;"interner Takt
SBHE~    pin           ;"16 Bit-Transfer
MEMR~    pin           ;"lesen
MEMW~    pin           ;"schreiben
IOR~     pin           ;"IO-lesen
IOW~     pin           ;"IO-schreiben
FREE~    pin           ;"interner Bus frei
RESET    pin           ;"allgemeiner RESET
"AEN      pin          ;
"Refresh Zyklus
A23      pin           ;"ungelatchte Adressen
A22      pin           ;"ungelatchte Adressen
A21      pin           ;"ungelatchte Adressen
A20      pin           ;"ungelatchte Adressen
A9       pin           ;"gelatchte Adressen
A8       pin           ;"gelatchte Adressen
A7       pin           ;"gelatchte Adressen
A6       pin           ;"gelatchte Adressen
A5       pin           ;"gelatchte Adressen
A4       pin           ;"gelatchte Adressen
A3       pin           ;"gelatchte Adressen
A2       pin           ;"gelatchte Adressen
A1       pin           ;"gelatchte Adressen
A0       pin           ;"gelatchte Adressen

"Ausg nge
"-----------------------------------------------------------------------
"Bussignale
IOCS16~  pin     istype 'invert';"Identifizierung einer 16 Bit-Karte
MEMCS16~ pin     istype 'invert';"Identifizierung einer 16 Bit-Karte
ZEROWS~  pin     istype 'reg,buffer';"Ohne Wartezyklen
WAIT~    pin     istype 'reg,invert';"Wartezyklen einf gen

"Steuerung der Bustreiber
"------------------------------------------------------------
DIN_EN~  pin     istype 'reg,invert';"internen Datentreiber freigeben
DEX_EN~  pin     istype 'reg,invert';"externen Datentreiber freigeben
DIN_LAT  pin     istype 'reg,buffer';"interne Daten latchen
DEX_LAT  pin     istype 'reg,buffer';"externe Daten latchen
ADR_LAT  pin     istype 'reg,buffer';"externe Adressen speichern
ADR_INC  pin     istype 'reg,invert';"interne Adressen inkrementieren

"interner Bus
"------------------------------------------------------------
WR_INT~  pin     istype 'reg,invert' ;"internes Schreiben
RD_INT~  pin     istype 'reg,invert' ;"internes Lesen
```

```
"interne Merker
"--------------------------------------------------------------
MEM_WR_ZY   node   istype 'com' ;"Speicher Schreibzugriff
MEM_RD_ZY   node   istype 'com' ;"Speicher Lesezugriff
IOSEL       node   istype 'com' ;"I/O Zugriff
SEL~        node   istype 'com' ;"Board select
READY       node    istype 'reg,buffer' ;"interner Zyklus abgeschlossen

"Vereinbarungen
"--------------------------------------------------------------
H,L,C,X,Z   = 1, 0, .C., .X.,.Z.;
IOA         = ^h30                   ;"Adrealage fuer IO-Adresse
MA          = ^hE                    ;"Adrealage fuer Memory
MADRES      = [A23,A22,A21,A20]      ;"Memory-Adresse
IOADRES     = [A9,A8,A7,A6,A5,A4];"IO-Adresse
RD_REQUEST = ( !WAIT~.fb & !DEX_LAT.fb )    ;"fuer interne Maschine
WR_REQUEST = ( !WAIT~.fb &  DEX_LAT.fb )    ;
IO_WR_ZY    = (IOSEL & !IOW~)      ;"IO Write Zyklus
IO_RD_ZY    = (IOSEL & !IOR~)      ;"IO Read Zyklus

AMDMACH property 'GROUP A WR_INT~'   ;
AMDMACH property 'GROUP A RD_INT~'   ;
AMDMACH property 'GROUP A DIN_EN~'   ;
AMDMACH property 'GROUP A READY'     ;
AMDMACH property 'GROUP A ADR_INC'   ;
AMDMACH property 'GROUP A DIN_LAT'   ;
AMDMACH property 'GROUP B ZEROWS~'   ;
"MDMACH property 'GROUP B WAIT~'     ;WAIT~ mag der Fitter nicht !?!
AMDMACH property 'GROUP B DEX_EN~'   ;
AMDMACH property 'GROUP B DEX_LAT'   ;
AMDMACH property 'GROUP B ADR_LAT'   ;
AMDMACH property 'GROUP A IOCS16~'   ;
AMDMACH property 'GROUP A IOSEL'     ;
AMDMACH property 'GROUP B SEL~'      ;
AMDMACH property 'GROUP B MEMCS16~'  ;
AMDMACH property 'GROUP B MEM_WR_ZY' ;
AMDMACH property 'GROUP B MEM_RD_ZY' ;

"Zust nde des externen Businterfaces
"--------------------------------------------------------------
EXT_ZYK    = [ZEROWS~,
                WAIT~,
                   DEX_EN~,
                      DEX_LAT,
                         ADR_LAT];
EXT_FREI   = [0 ,1 ,1 ,0 ,0 ];"Ruhezustand
RD_R       = [0 ,1 ,0 ,0 ,0 ];"Register lesen
RD_R_WAIT = [1 ,0 ,1 ,0 ,0 ];"Register lesen
WR_R       = [0 ,1 ,1 ,1 ,0 ];"Register schreiben
WR_R_WAIT = [1 ,0 ,1 ,1 ,0 ];"Register schreiben

RD_M       = [0 ,1 ,0 ,0 ,1 ];"Register lesen
RD_M_WAIT = [1 ,0 ,1 ,0 ,1 ];"Register lesen
WR_M       = [0 ,1 ,1 ,1 ,1 ];"Register schreiben
WR_M_WAIT = [1 ,0 ,1 ,1 ,1 ];"Register schreiben

"Zust nde des internen Businterfaces
"--------------------------------------------------------------
INT_ZYK    = [DIN_EN~,
                DIN_LAT ,
```

```
                WR_INT~,
                   RD_INT~,
                      READY,
                         ADR_INC];
INT_FREI   = [ 1 ,0 ,1 ,1 ,0 ,1 ];"Ruhezustand
INT_WRITE = [ 0 ,0 ,1 ,1 ,0 ,1 ];"Daten freigeben
WRITEINT   = [ 0 ,0 ,0 ,1 ,1 ,0 ];"intern schreiben
INT_READ   = [ 1 ,0 ,1 ,0 ,0 ,1 ];"intern lesen
LATCHINT   = [ 1 ,1 ,1 ,0 ,1 ,0 ];"intern latchen

TEST_ZYK    = [ZEROWS~,
                WAIT~,
                   DEX_EN~,
                      DEX_LAT ,
                         ADR_LAT,
                            DIN_EN~,
                               DIN_LAT ,
                                     WR_INT~,
                                        RD_INT~,
                                           READY,
                                              ADR_INC];

state_diagram  EXT_ZYK
"---------------------------------------------------------------------
state EXT_FREI  :

        if MEM_WR_ZY then WR_M else
        if MEM_RD_ZY then RD_M else
        if IO_RD_ZY  then RD_R else
        if IO_WR_ZY  then WR_R else EXT_FREI;

state RD_R :
        if !(IO_RD_ZY) then RD_R_WAIT else RD_R;

state RD_R_WAIT :
        if READY then EXT_FREI else RD_R_WAIT;

state WR_R :
        if !(IO_WR_ZY) then WR_R_WAIT else WR_R;

state WR_R_WAIT :
        if READY then EXT_FREI else WR_R_WAIT;

state RD_M :
        if !(MEM_RD_ZY) then RD_M_WAIT else RD_M;

state RD_M_WAIT :
        if READY then EXT_FREI else RD_M_WAIT;

state WR_M :
        if !(MEM_WR_ZY) then WR_M_WAIT else WR_M;

state WR_M_WAIT :
        if READY then EXT_FREI else WR_M_WAIT;

state_diagram  INT_ZYK
"---------------------------------------------------------------------
state INT_FREI  :

        if (RD_REQUEST & !FREE~) then INT_READ  else
        if (WR_REQUEST & !FREE~) then INT_WRITE else INT_FREI;
```

```
state INT_WRITE  :
        goto WRITEINT;

state WRITEINT:
        goto INT_FREI;

state INT_READ :
        goto LATCHINT;

state LATCHINT:
        goto INT_FREI;

equations
"-----------------------------------------------------------------------
[EXT_ZYK].c  = EXTAKT   ;"
[EXT_ZYK].ar = RESET    ;"
[EXT_ZYK].ap = L        ;"
[INT_ZYK].c  = INTAKT   ;"
[INT_ZYK].ar = RESET    ;"
[INT_ZYK].ap = L        ;"

"Gleichungen fuer Adressdekoder
"-----------------------------------------------------------------------
MEM_WR_ZY = ((MADRES == MA)  & !MEMW~)     ;"Memory Write Zyklus
MEM_RD_ZY = ((MADRES == MA)  & !MEMR~)     ;"Memory Read Zyklus
IOSEL     = ((IOADRES == IOA) & (!IOR~ # !IOW~));"IO Zyklus

MEMCS16~ =!(!SBHE~          & (MADRES  == MA))  ;"
IOCS16~  =!(!SBHE~          & (IOADRES == IOA)) ;"
SEL~      =! (MEM_RD_ZY # MEM_WR_ZY # IOSEL);

IOCS16~ .oe = !SEL~.fb          ;"open Colector Simulation
MEMCS16~.oe = !SEL~.fb          ;"open Colector Simulation
ZEROWS~ .oe = !SEL~.fb          ;"Nur antworten, wenn gefragt
WAIT~   .oe = !SEL~.fb          ;"Nur antworten, wenn gefragt
WR_INT~ .oe = !FREE~            ;"Nur wenn interner Bus frei ist
RD_INT~ .oe = !FREE~            ;"Nur wenn interner Bus frei ist
```

```
test_vectors
"---------------------------------------------------------------------------------------
"Test der Adreadekoder
([IOADRES
     ,MADRES
        ,SBHE~
           ,IOR~
              ,IOW~
                 ,MEMR~
                     ,MEMW~]
                        ->[IOCS16~
                            ,IOSEL
                               ,MEMCS16~
                                  ,MEM_WR_ZY
                                     ,MEM_RD_ZY])

 [IOA, X, 0, 0, 1, 1, 1]->[0, 1, 1, 0, 0];" 1
 [IOA, X, 1, 0, 1, 1, 1]->[1, 1, 1, 0, 0];" 2
 [IOA, X, 0, 1, 0, 1, 1]->[0, 1, 1, 0, 0];" 3
 [IOA, X, 1, 1, 0, 1, 1]->[1, 1, 1, 0, 0];" 4
 [  X,MA, 0, 1, 1, 0, 1]->[1, 0, 0, 0, 1];" 5
 [  X,MA, 0, 1, 1, 1, 0]->[1, 0, 0, 1, 0];" 6
 [  X, X, 1, 1, 1, 1, 1]->[Z, 0, Z, 0, 0];" 7
 [  X, X, 1, 1, 1, 1, 1]->[Z, 0, Z, 0, 0];" 8
 [ X, X, 1, 1, 1, 1, 1]->[Z, 0, Z, 0, 0];" 9
 [  X, X, 1, 1, 1, 1, 1]->[Z, 0, Z, 0, 0];"10

 test_vectors
"-----------------------------------------------------------------------
" berpr fen der beiden Zustandsmaschinen
"Zun chst Register lesen
([EXTAKT,INTAKT,RESET,SBHE~,IOADRES,IOR~,IOW~,MADRES,MEMR~,MEMW~,FREE~]
->TEST_ZYK)
"             I                            D A   D       A
"   I E       O        M             Z   D E D D I W R   D
"   N X R     A        A             E   E X R I N R D R R
"   T T E S   D        D  M M F      R W X _ _ N _ _ _ E _
"   A A S B   R  I I   R  E E R      O A _ L L _ L I I A I
"   K K E H   E  O O   E  M M E      W I E A A E A N N D N
"   T T T E   S  R W   S  R W E      S T N T T N T T T Y C
"         ~      ~ ~      ~ ~ ~      ~ ~ ~     ~   ~ ~
 [  C,C,1,X,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"11 Reset-Zustand
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"12 Beide Maschinen frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"13
"Simulation: Einfaches Register lesen
 [ C,C,0,0,IOA ,0,1,  X ,1,1,1]-> [ 0,1,0,0,0,1,0,Z,Z,0,1];"14 Lesezugriff auf Register
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"15 Warten bis intern frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,0,1,0,1,0,0,1];"16 Internes Lesen
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,0,1,1,1,0,1,0];"17 Datum intern latchen
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"18 Beide Maschinen frei
"Simulation: Register lesen (2 Takte), dann erneut lesen mit warten
"             I                            D A   D       A
"   I E       O        M             Z   D E D D I W R   D
"   N X R     A        A             E   E X R I N R D R R
"   T T E S   D        D  M M F      R W X _ _ N _ _ _ E _
"   A A S B   R  I I   R  E E R      O A _ L L _ L I I A I
"   K K E H   E  O O   E  M M E      W I E A A E A N N D N
"   T T T E   S  R W   S  R W E      S T N T T N T T T Y C
"         ~      ~ ~      ~ ~ ~      ~ ~ ~     ~   ~ ~
```

```
 [ C,C,0,0,IOA ,0,1,  X ,1,1,1]-> [ 0,1,0,0,0,1,0,Z,Z,0,1];"19 Lesezugriff auf Register
  [  C,C,0,0,IOA ,0,1,  X ,1,1,1]-> [ 0,1,0,0,0,1,0,Z,Z,0,1];"20 dauert etwas l nger
  [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"21 Warten bis intern frei
 [ C,C,0,1,IOA ,0,1,  X ,1,1,0]-> [ 1,0,1,0,0,1,0,1,0,0,1];"22 Internes Lesen, Bus will
schon wieder
  [  C,C,0,1,IOA ,0,1,  X ,1,1,0]-> [ 1,0,1,0,0,1,1,1,0,1,0];"23 Datum intern latchen
  [  C,C,0,0,IOA ,0,1,  X ,1,1,1]-> [ 0,1,1,0,0,1,0,Z,Z,0,1];"24 Bus muss noch warten
  [  C,C,0,1,IOA ,0,1,  X ,1,1,1]-> [ 0,1,0,0,0,1,0,Z,Z,0,1];"25 Warten bis intern frei
 [ C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,0,1,0,1,1,0,1];"26 warten auf internes Lesen
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,0,1,0,1,0,0,1];"27 Internes Lesen
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,0,1,1,1,0,1,0];"28 Datum intern latchen
  [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"29 Beide Maschinen frei
"-------------------------------------------------------------------------------
"Jetzt Register schreiben
"             I                            D A   D       A
"   I E       O        M             Z   D E D D I W R   D
"   N X R     A        A             E   E X R I N R D R R
"   T T E S   D        D  M M F      R W X _ _ N _ _ _ E _
"   A A S B   R  I I   R  E E R      O A _ L L _ L I I A I
"   K K E H   E  O O   E  M M E      W I E A A E A N N D N
"   T T T E   S  R W   S  R W E      S T N T T N T T T Y C
"         ~      ~ ~      ~ ~ ~      ~ ~ ~     ~   ~ ~
  [  C,C,1,X,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"30 Reset-Zustand
  [  C,C,1,X,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"31 Reset-Zustand
  [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"32 Beide Maschinen frei
  [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"33
"Simulation: Einfaches Register schreiben
 [ C,C,0,0,IOA ,1,0,  X ,1,1,1]-> [ 0,1,1,1,0,1,0,Z,Z,0,1];"34 Schreibzugriff auf Reg-
ister
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,1,0,1,0,Z,Z,0,1];"35 Warten bis intern frei
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,0,0,0,1,1,0,1];"36
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,0,0,0,0,1,1,0];"37 Internes Schreiben
  [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"38 Beide Maschinen frei
"Simulation: Register schreiben (2 Takte), dann erneut schreiben mit warten
"             I                            D A   D       A
"   I E       O        M             Z   D E D D I W R   D
"   N X R     A        A             E   E X R I N R D R R
"   T T E S   D        D  M M F      R W X _ _ N _ _ _ E _
"   A A S B   R  I I   R  E E R      O A _ L L _ L I I A I
"   K K E H   E  O O   E  M M E      W I E A A E A N N D N
"   T T T E   S  R W   S  R W E      S T N T T N T T T Y C
"         ~      ~ ~      ~ ~ ~      ~ ~ ~     ~   ~ ~
 [ C,C,0,0,IOA ,1,0,  X ,1,1,1]-> [ 0,1,1,1,0,1,0,Z,Z,0,1];"39 Schreibzugriff auf Reg-
ister
  [  C,C,0,0,IOA ,1,0,  X ,1,1,1]-> [ 0,1,1,1,0,1,0,Z,Z,0,1];"40 dauert etwas l nger
 [ C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,1,0,1,0,Z,Z,0,1];"41 Warten bis intern frei
  [  C,C,0,1,IOA ,1,0,  X ,1,1,0]-> [ 1,0,1,1,0,0,0,1,1,0,1];"42 Bus will schon wieder
  [  C,C,0,1,IOA ,1,0,  X ,1,1,0]-> [ 1,0,1,1,0,0,0,0,1,1,0];"43
  [  C,C,0,0,IOA ,1,0,  X ,1,1,1]-> [ 0,1,1,0,0,1,0,Z,Z,0,1];"44 Bus muss noch warten
 [  C,C,0,1,IOA ,1,0,  X ,1,1,1]-> [ 0,1,1,1,0,1,0,Z,Z,0,1];"45 Warten bis intern frei
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,0,1,0,1,1,0,1];"46 warten auf internes
schreiben
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,0,0,0,1,1,0,1];"47
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,0,0,0,0,1,1,0];"48 Internes schreiben
  [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"49 Beide Maschinen frei

test_vectors
"Zun chst Speicher lesen
([EXTAKT,INTAKT,RESET,SBHE~,IOADRES,IOR~,IOW~,MADRES,MEMR~,MEMW~,FREE~]
```

```
->TEST_ZYK)
"           I                      D A  D   A
"   I E     O      M             Z   D E D D I W R  D
"   N X R   A      A             E   E X R I N R D R R
"   T T E S D      D  M M F      R W X _ _ N _ _ _ E _
"   A A S B R  I I R  E E R      O A _ L L _ L I I A I
"   K K E H E  O O E  M M E      W I E A A E A N N D N
"   T T T E S  R W S  R W E      S T N T T N T T T Y C
"       ~      ~ ~    ~ ~ ~      ~ ~ ~ ~ ~   ~ ~ ~
 [  C,C,1,X,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"11 Reset-Zustand
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"12 Beide Maschinen frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"13
"Simulation: Einfaches Speicher lesen
 [ C,C,0,0,  X ,1,1, MA ,0,1,1]-> [ 0,1,0,0,1,1,0,Z,Z,0,1];"14 Lesezugriff auf Speicher
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,1,1,0,Z,Z,0,1];"15 Warten bis intern frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,1,1,0,1,0,0,1];"16 Internes Lesen
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,1,1,1,1,0,1,0];"17 Datum intern latchen
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"18 Beide Maschinen frei
"Simulation: Speicher lesen (2 Takte), dann erneut lesen mit warten
"           I                      D A  D   A
"   I E     O      M             Z   D E D D I W R  D
"   N X R   A      A             E   E X R I N R D R R
"   T T E S D      D  M M F      R W X _ _ N _ _ _ E _
"   A A S B R  I I R  E E R      O A _ L L _ L I I A I
"   K K E H E  O O E  M M E      W I E A A E A N N D N
"   T T T E S  R W S  R W E      S T N T T N T T T Y C
"       ~      ~ ~    ~ ~ ~      ~ ~ ~       ~   ~ ~
 [ C,C,0,0,  X ,1,1, MA ,0,1,1]-> [ 0,1,0,0,1,1,0,Z,Z,0,1];"19 Lesezugriff auf Speicher
 [  C,C,0,0,  X ,1,1, MA ,0,1,1]-> [ 0,1,0,0,1,1,0,Z,Z,0,1];"20 dauert etwas l nger
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,1,1,0,Z,Z,0,1];"21 Warten bis intern frei
 [ C,C,0,1,  X ,1,1, MA ,0,1,0]-> [ 1,0,1,0,1,1,0,1,0,0,1];"22 Internes Lesen, Bus will
schon wieder
 [  C,C,0,1,  X ,1,1, MA ,0,1,0]-> [ 1,0,1,0,1,1,1,1,0,1,0];"23 Datum intern latchen
 [  C,C,0,0,  X ,1,1, MA ,0,1,1]-> [ 0,1,1,0,0,1,0,Z,Z,0,1];"24 Bus muss noch warten
 [  C,C,0,1,  X ,1,1, MA ,0,1,1]-> [ 0,1,0,0,1,1,0,Z,Z,0,1];"25 Warten bis intern frei
 [ C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,1,1,0,1,1,0,1];"26 warten auf internes Lesen
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,1,1,0,1,0,0,1];"27 Internes Lesen
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,0,1,1,1,1,0,1,0];"28 Datum intern latchen
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"29 Beide Maschinen frei
"-----------------------------------------------------------------------
"Jetzt Speicher schreiben
"           I                      D A  D   A
"   I E     O      M             Z   D E D D I W R  D
"   N X R   A      A             E   E X R I N R D R R
"   T T E S D      D  M M F      R W X _ _ N _ _ _ E _
"   A A S B R  I I R  E E R      O A _ L L _ L I I A I
"   K K E H E  O O E  M M E      W I E A A E A N N D N
"   T T T E S  R W S  R W E      S T N T T N T T T Y C
"       ~      ~ ~    ~ ~ ~      ~ ~ ~       ~   ~ ~
 [  C,C,1,X,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"30 Reset-Zustand
 [  C,C,1,X,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"31 Reset-Zustand
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"32 Beide Maschinen frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"33
"Simulation: Einfaches Speicher schreiben
 [  C,C,0,0,  X ,1,1, MA ,1,0,1]-> [ 0,1,1,1,1,1,0,Z,Z,0,1];"34 Schreibzugriff auf
Speicher
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,1,1,1,0,Z,Z,0,1];"35 Warten bis intern frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,1,0,0,1,1,0,1];"36
```

```
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,1,0,0,0,1,1,0];"37 Internes Schreiben
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"38 Beide Maschinen frei
"Simulation: Speicher schreiben (2 Takte), dann erneut schreiben mit warten
"           I                        D A  D    A
"   I E     O       M        Z   D E D D I W R  D
"   N X R   A       A        E   E X R I N R D R R
"   T T E S D       D M M F  R W X _ _ N _ _ _ E _
"   A A S B R  I I  R E E R  O A _ L L _ L I I A I
"   K K E H E  O O  E M M E  W I E A A E A N N D N
"   T T T E S  R W  S R W E  S T N T T N T T Y C
"        ~    ~  ~        ~    ~ ~     ~  ~ ~
  [  C,C,0,0,  X ,1,1, MA ,1,0,1]-> [ 0,1,1,1,1,1,0,Z,Z,0,1];"39 Schreibzugriff auf
Speicher
 [  C,C,0,0,  X ,1,1, MA ,1,0,1]-> [ 0,1,1,1,1,1,0,Z,Z,0,1];"40 dauert etwas l nger
 [ C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,1,1,1,0,Z,Z,0,1];"41 Warten bis intern frei
 [  C,C,0,1,  X ,1,1, MA ,1,0,0]-> [ 1,0,1,1,1,0,0,1,1,0,1];"42 Bus will schon wieder
 [  C,C,0,1,  X ,1,1, MA ,1,0,0]-> [ 1,0,1,1,1,0,0,0,1,1,0];"43
 [  C,C,0,0,  X ,1,1, MA ,1,0,1]-> [ 0,1,1,0,0,1,0,Z,Z,0,1];"44 Bus muss noch warten
 [ C,C,0,1,  X ,1,1, MA ,1,0,1]-> [ 0,1,1,1,1,1,0,Z,Z,0,1];"45 Warten bis intern frei
  [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,1,1,0,1,1,0,1];"46 warten auf internes
schreiben
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,1,0,0,1,1,0,1];"47
 [  C,C,0,1,  X ,1,1,  X ,1,1,0]-> [ Z,Z,1,1,1,0,0,0,1,1,0];"48 Internes Schreiben
 [  C,C,0,1,  X ,1,1,  X ,1,1,1]-> [ Z,Z,1,0,0,1,0,Z,Z,0,1];"49 Beide Maschinen frei

end
```

FITTER FILE: MACH110

Note: This file has been condensed in order to save trees.

```
AMD MACH FITR - MARKET RELEASE (1-24-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990

Flags Used:           Unplace=False          Max Packing=True
Flags Used:      Expand Small=False          Expand All=False

*************************************************************
     Mach PLD Fitter - v 1.46            ISA-BUS-INTERFACE
*************************************************************

*** Timing Analysis for Signals

 Parameter   Min   Max        Signal List (Those having Max delay.)
      Tpd     1     3         IOCS16~        MEMCS16~
      Tsu     1     2         ZEROWS~          WAIT~           DEX_EN~
                              DEX_LAT        ADR_LAT
      Tco     0     0          WAIT~         DEX_LAT         ZEROWS~
                              DEX_EN~        ADR_LAT         DIN_EN~
                              DIN_LAT        ADR_INC
      Tcr     1     1         ZEROWS~          WAIT~         DIN_EN~
                              DIN_LAT        DEX_LAT         ADR_LAT
                              ADR_INC          READY

 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Register clock to combinatorial output
 Tcr - Register thru combinatorial logic to setup
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                Available          Used          Remaining
        Clocks:     2                2              0
          Pins:    38               31              7       ->    81%
     I/O Macro:    32               12             20
   Total Macro:    32               17             15
 Product Terms:   128               78             20       ->    84%

MACH-PLD Resource Checks OK!

Partitioning Design into Blocks...

*** Last Equations Placed in Blocks

Weakly -
Assign -

*** Block Partitioning Results

              Array     Macros     # I/O     Buried    Product     Signal
              Inputs    Remain     Macro     Logic     Terms       Fanout
   Block-> A   20          8         6         2         48           8
   Block-> B   20          7         6         3         60          11

*** Block Signal List

Block-> A          ADR_INC          RD_INT~          WR_INT~          DIN_LAT
                   DIN_EN~            READY            IOSEL          IOCS16~
```

```
Block-> B            WAIT~            ADR_LAT              DEX_EN~         ZEROWS~
                   DEX_LAT               SEL~         MEM_RD_ZY         MEM_WR_ZY
                  MEMCS16~

|> INFORMATION F050 - Device Utilization....... *: 84 %

 *** Feedback Map - ISA-BUS-INTERFACE

 Gbl Inp .--.       I/O  .--+--A--+--. I/O               I/O .--+--B--+--.  I/O
         | 0|            | 0|     |21| IOW~          A23 : 0|     |21| IOW~
         | 1|  /RD_INT~  1|     |20| IOR~         MEMW~ : 1|     |20| IOR~
         | 2|   /WAIT~ : 2|     |19| A9          /WAIT~ : 2|     |19| MEMR~
         | 3|  /ADR_INC  3|     |18|               A22 : 3|     |18| A20
         | 4|        A8 : 4|     |17| RESET          A21 : 4|     |17| RESET
         | 5|  /DIN_EN~  5|     |16| SBHE~       /DEX_EN~  5|     |16| SBHE~
         '--'  DEX_LAT : 6|     |15: DIN_LAT     DEX_LAT : 6|     |15|
                   A7 : 7|     |14: FREE~                | 7|     |14: MEM_RD_
               /WR_INT~  8|     |13: SEL~        ZEROWS~ : 8|     |13: SEL~
                   A6 : 9|     |12: READY         IOSEL : 9|     |12: READY
                   A5 :10|     |11: A4          ADR_LAT :10|     |11: MEM_WR_
                         '--+-u--u+--'                      '--+-u--u+--'

*** Logic Map - ISA-BUS-INTERFACE

 Gbl Inp .--.       I/O  .--+--A--+--.  I/O              I/O  .--+--B--+--.  I/O
   SBHE~| 0|  /IOCS16~  0| 1    |21|              /MEMCS16  0| 1    |21|
   RESET| 1|  /RD_INT~  1| 8    |20|                       | 1| *    |20|
  INTAKT| 2|            | 2| *    |19|               /WAIT~ | 2|12    |19|
    IOR~| 3|  /ADR_INC  3| 7    |18|                      | 3| *    |18|
    IOW~| 4|            | 4| *    |17|                      | 4| *    |17|
  EXTAKT| 5|  /DIN_EN~  5| 8    |16|              /DEX_EN~  5| 9    |16|
         '--'           | 6| *   1|15| DIN_LAT      DEX_LAT | 6| 4   .|15|
                        | 7| .   .|14|                      | 7| *   1|14| MEM_RD_
              /WR_INT~  8| 8   .|13|              ZEROWS~ | 8| 7   1|13| SEL~
                IOSEL | 9| 2   2|12| READY                 | 9| *   *|12|
                      |10| *   .|11|              ADR_LAT |10| 5   1|11| MEM_WR_
                       '--+-u--u+--'                       '--+-u--u+--'
```

```
*** Pin Map - ISA-BUS-INTERFACE



                                /IOCS16~
                             /RD_INT~  |        .
                             MEMW~  |  |        | A6
                       /ADR_INC  |  |  |        |  | A5
                          A7  |  |  |  |        |  |  | A4
                           |  |  |  |  |        |  |  |  |
                    .------'--'--'--'--'--o-----'--'--'--'---.
                    |                        4  4  4  4  4   |
                    |      6  5  4  3  2  1  4  3  2  1  0   |
            /DIN_EN~| 7                                   39|MEMR~
                    | 8                   G  V            38|ADR_LAT
                  A8| 9                   n  c            37|FREE~
               SBHE~|10                   d  c            36|ZEROWS~
               RESET|11                                   35|EXTAKT
                 Gnd|12            MACH-110               34| Gnd
              INTAKT|13                                   33|IOW~
            /WR_INT~|14                V  G               32|IOR~
                    |15                c  n               31|A23
                    |16                c  d               30|DEX_LAT
                    |17                                   29|/DEX_EN~
                    |    1  1  2  2  2  2  2  2  2  2  2     |
                    |    8  9  0  1  2  3  4  5  6  7  8     |
                    '----.--.--.--.--------.--.--.--.--.-----'
                         |  |  |  |        |  |  |  |  |
                        A9  |  |  |        |  |  |  | A22
                            '  |  |        |  |  | A21
                               '  |        |  | /WAIT~
                           DIN_LAT         |  A20
                                          /MEMCS16~

The Design Doc is  stored in ===> isa.Rpt
The Jedec Data is  stored in ===> isa.Jed
The Placements are stored in ===> isa.Plc
The  Fuse Plot is  stored in ===> isa.Xpt

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: isa
```

PLACEMENT FILE: MACH110

```
; Flags Used:             Unplace=False         Max Packing=True
; Flags Used:        Expand Small=False          Expand All=False

; FITR generated placements

Pin    35          EXTAKT  ; Inp          ; I  5
Pin    13          INTAKT  ; Inp          ; I  2
Pin    10           SBHE~  ; Inp          ; I  0
Pin    39           MEMR~  ; Inp          ; B 11
Pin     4           MEMW~  ; Inp          ; A  2
Pin    32            IOR~  ; Inp          ; I  3
Pin    33            IOW~  ; Inp          ; I  4
Pin    37           FREE~  ; Inp          ; B  9
Pin    11           RESET  ; Inp          ; I  1
Pin    31             A23  ; Inp          ; B  7
Pin    28             A22  ; Inp          ; B  4
Pin    27             A21  ; Inp          ; B  3
Pin    25             A20  ; Inp          ; B  1
Pin    18              A9  ; Inp          ; A 12
Pin     9              A8  ; Inp          ; A  7
Pin     6              A7  ; Inp          ; A  4
Pin    42              A6  ; Inp          ; B 14
Pin    41              A5  ; Inp          ; B 13
Pin    40              A4  ; Inp          ; B 12
Pin     2        /IOCS16~    Comb         ; A  0
Pin    24       /MEMCS16~    Comb         ; B  0
Node   26       ZEROWS~_I    Reg          ; B  8
Node   20        /WAIT~_I    Reg          ; B  2
Node    7      /DIN_EN~_I    Reg          ; A  5
Node   23      /DEX_EN~_I    Reg          ; B  5
Node   17       DIN_LAT_I    Reg          ; A 15
Node   24       DEX_LAT_I    Reg          ; B  6
Node   28       ADR_LAT_I    Reg          ; B 10
Node    5      /ADR_INC_I    Reg          ; A  3
Node   10      /WR_INT~_I    Reg          ; A  8
Node    3      /RD_INT~_I    Reg          ; A  1
Node   29       MEM_WR_ZY    Comb         ; B 11
Node   32       MEM_RD_ZY    Comb         ; B 14
Node   11           IOSEL    Comb         ; A  9
Node   31            SEL~    Comb         ; B 13
Node   14           READY    Reg          ; A 12
Pin    26          /WAIT~    Reg          ; B  2
Pin    30         DEX_LAT    Reg          ; B  6
Pin    36         ZEROWS~    Reg          ; B  8
Pin    29        /DEX_EN~    Reg          ; B  5
Pin    38         ADR_LAT    Reg          ; B 10
Pin     7        /DIN_EN~    Reg          ; A  5
Pin    21         DIN_LAT    Reg          ; A 15
Pin    14        /WR_INT~    Reg          ; A  8
Pin     3        /RD_INT~    Reg          ; A  1
Pin     5        /ADR_INC    Reg          ; A  3

; Group Mach_Seg_A ADR_INC RD_INT~ WR_INT~ DIN_LAT DIN_EN~ READY IOSEL IOCS16~
; Group Mach_Seg_B WAIT~ ADR_LAT DEX_EN~ ZEROWS~ DEX_LAT SEL~ MEM_RD_ZY MEM_WR_ZY
MEMCS16~
```

DESIGN FILE: MACH210

```
module ISA
title   'ISA-BUS-INTERFACE'
declarations
"---------------------------------------------------------------------------

ISA      device 'mach210';

"Eing nge
"---------------------------------------------------------------------------
EXTAKT   pin            ;"Bustakt
INTAKT   pin            ;"interner Takt
SBHE~    pin            ;"16 Bit-Transfer
MEMR~    pin            ;"lesen
MEMW~    pin            ;"schreiben
IOR~     pin            ;"IO-lesen
IOW~     pin            ;"IO-schreiben
FREE~    pin            ;"interner Bus frei
RESET    pin            ;"allgemeiner RESET
AEN      pin            ;"Refresh Zyklus
A23      pin            ;"ungelatchte Adressen
A22      pin            ;"ungelatchte Adressen
A21      pin            ;"ungelatchte Adressen
A20      pin            ;"ungelatchte Adressen
A9       pin            ;"gelatchte Adressen
A8       pin            ;"gelatchte Adressen
A7       pin            ;"gelatchte Adressen
A6       pin            ;"gelatchte Adressen
A5       pin            ;"gelatchte Adressen
A4       pin            ;"gelatchte Adressen
A3       pin            ;"gelatchte Adressen
A2       pin            ;"gelatchte Adressen
A1       pin            ;"gelatchte Adressen
A0       pin            ;"gelatchte Adressen

"Ausg nge
"---------------------------------------------------------------------------
"Bussignale
IOCS16~  pin   istype 'invert'     ;"Identifizierung einer 16 Bit-Karte
MEMCS16~ pin   istype 'invert'     ;"Identifizierung einer 16 Bit-Karte
ZEROWS~  pin   istype 'reg,buffer';"Ohne Wartezyklen
WAIT~    pin   istype 'reg,invert';"Wartezyklen einf gen

"Steuerung der Bustreiber
"---------------------------------------------------------
DIN_EN~  pin   istype 'reg,buffer';"internen Datentreiber freigeben
DEX_EN~  pin   istype 'reg,invert';"externen Datentreiber freigeben
DIN_LAT  pin   istype 'reg,buffer';"interne Daten latchen
DEX_LAT  pin   istype 'reg,buffer';"externe Daten latchen
ADR_LAT  pin   istype 'reg,buffer';"externe Adressen speichern
ADR_INC  pin   istype 'reg,buffer';"interne Adressen inkrementieren

"interner Bus
"-----------------------------------------------------------
WR_INT~  pin   istype 'reg,buffer' ;"internes Schreiben
RD_INT~  pin   istype 'reg,buffer' ;"internes Lesen

"interne Signale
"-----------------------------------------------------------
```

```
MEM_WR_ZY  node istype 'com'      ;"Speicher Schreibzugriff
MEM_RD_ZY  node istype 'com'      ;"Speicher Lesezugriff
IO_WR_ZY   node istype 'com'      ;"IO Schreibzugriff
IO_RD_ZY   node istype 'com'      ;"IO Lesezugriff
SEL~       node istype 'com'      ;"Board select
READY      pin  istype 'reg,invert' ;"interner Zyklus abgeschlossen

"Vereinbarungen
"-----------------------------------------------------------
H,L,C,X,Z  = 1, 0, .C., .X.,.Z.;
IOA        = ^h300                ;"Adrealage fuer IO-Adresse
MA         = ^hE                  ;"Adrealage fuer Memory
MADRES     = [A23,A22,A21,A20]    ;"Memory-Adresse
IOADRES    = [A9,A8,A7,A6,A5,A4,A3,A2,A1,A0];"IO-Adresse
RD_REQUEST = ( !WAIT~.fb & !DEX_LAT.fb ) ;"fuer interne Maschine
WR_REQUEST = ( !WAIT~.fb &  DEX_LAT.fb ) ;

"Zust nde des externen Businterfaces
"-----------------------------------------------------------
EXT_ZYK    = [ZEROWS~,
                WAIT~,
                   DEX_EN~,
                      DEX_LAT,
                         ADR_LAT];
EXT_FREI   = [0 ,1 ,1 ,0 ,0 ];"Ruhezustand
RD_R       = [0 ,1 ,0 ,0 ,0 ];"Register lesen
RD_R_WAIT  = [1 ,0 ,1 ,0 ,0 ];"Register lesen
WR_R       = [0 ,1 ,1 ,1 ,0 ];"Register schreiben
WR_R_WAIT  = [1 ,0 ,1 ,1 ,0 ];"Register schreiben

RD_M       = [0 ,1 ,0 ,0 ,1 ];"Register lesen
RD_M_WAIT  = [1 ,0 ,1 ,0 ,1 ];"Register lesen
WR_M       = [0 ,1 ,1 ,1 ,1 ];"Register schreiben
WR_M_WAIT  = [1 ,0 ,1 ,1 ,1 ];"Register schreiben

"Zust nde des internen Businterfaces
"-----------------------------------------------------------
INT_ZYK    = [DIN_EN~,
                DIN_LAT ,
                   WR_INT~,
                      RD_INT~,
                         READY,
                            ADR_INC];
INT_FREI   = [ 1 ,1 ,1 ,1 ,0 ,1 ];"Ruhezustand
INT_WRITE  = [ 0 ,1 ,1 ,1 ,0 ,1 ];"Daten freigeben
WRITEINT   = [ 0 ,1 ,0 ,1 ,1 ,0 ];"intern schreiben
INT_READ   = [ 1 ,0 ,1 ,0 ,0 ,1 ];"intern lesen
LATCHINT   = [ 1 ,1 ,1 ,0 ,1 ,0 ];"intern latchen

TEST_ZYK   = [ZEROWS~,WAIT~,DEX_EN~,DEX_LAT,ADR_LAT,DIN_EN~,DIN_LAT
             ,WR_INT~,RD_INT~,READY,ADR_INC];

state_diagram  EXT_ZYK
"-----------------------------------------------------------
state EXT_FREI  :

        if MEM_WR_ZY then WR_M else
        if MEM_RD_ZY then RD_M else
        if IO_RD_ZY  then RD_R else
        if IO_WR_ZY  then WR_R else EXT_FREI;
```

```
state RD_R :
        if !(IO_RD_ZY) then RD_R_WAIT else RD_R;

state RD_R_WAIT :
        if READY then EXT_FREI else RD_R_WAIT;

state WR_R :
        if !(IO_WR_ZY) then WR_R_WAIT else WR_R;

state WR_R_WAIT :
        if READY then EXT_FREI else WR_R_WAIT;

state RD_M :
        if !(MEM_RD_ZY) then RD_M_WAIT else RD_M;

state RD_M_WAIT :
        if READY then EXT_FREI else RD_M_WAIT;

state WR_M :
        if !(MEM_WR_ZY) then WR_M_WAIT else WR_M;

state WR_M_WAIT :
        if READY then EXT_FREI else WR_M_WAIT;

state_diagram  INT_ZYK
"_____________________________________________________________________
state INT_FREI  :

        if (RD_REQUEST & !FREE~) then INT_READ  else
        if (WR_REQUEST & !FREE~) then INT_WRITE else INT_FREI;

state INT_WRITE  :
        goto WRITEINT;

state WRITEINT:
        goto INT_FREI;

state INT_READ :
        goto LATCHINT;

state LATCHINT:
        goto INT_FREI;

equations
"_____________________________________________________________________
[EXT_ZYK].c  = EXTAKT   ;" Laeuft mit Bustakt
[INT_ZYK].c  = INTAKT   ;" Laeuft mit internem Takt
[EXT_ZYK].ar = RESET    ;" Ruecksetzen
[INT_ZYK].ap = RESET    ;" Vorsetzen fuer interne Maschine
[EXT_ZYK].ap = L        ;" nicht setzen
[INT_ZYK].ap = L        ;" nicht setzen

"Gleichungen fuer Adressdekoder
"_____________________________________________________________________
MEM_WR_ZY = ((MADRES == MA) & !MEMW~ & !AEN ) ;"Memory Write Zyklus
MEM_RD_ZY = ((MADRES == MA) & !MEMR~ & !AEN ) ;"Memory Read Zyklus
IO_WR_ZY  = ((IOADRES == IOA) & !IOW~ & !AEN ) ;"IO Write Zyklus
IO_RD_ZY  = ((IOADRES == IOA) & !IOR~ & !AEN ) ;"IO Read Zyklus

MEMCS16~ = !(!SBHE~ & !AEN & (MADRES  == MA))   ;"Rueckmeldung 16-Bit Karte
IOCS16~  = !(!SBHE~ & !AEN & (IOADRES == IOA))  ;"Rueckmeldung 16-Bit Karte
SEL~     = !(MEM_RD_ZY # MEM_WR_ZY # IO_RD_ZY # IO_WR_ZY );
```

```
IOCS16~ .oe = !IOCS16~.fb        ;"open Colector Simulation
MEMCS16~.oe = !MEMCS16~.fb       ;"open Colector Simulation
ZEROWS~ .oe = !SEL~.fb           ;"Nur antworten, wenn gefragt
WAIT~   .oe = !SEL~.fb           ;"Nur antworten, wenn gefragt
WR_INT~ .oe = !FREE~             ;"Nur wenn interner Bus frei ist
RD_INT~ .oe = !FREE~             ;"Nur wenn interner Bus frei ist

test_vectors
"------------------------------------------------------------------
"Test der Adreadekoder
([IOADRES
     ,MADRES
        ,SBHE~
           ,IOR~
              ,IOW~
                 ,MEMR~
                     ,MEMW~
                        ,AEN]
                           ->[IOCS16~
                               ,IO_WR_ZY
                                  ,IO_RD_ZY
                                     ,MEMCS16~
                                        ,MEM_WR_ZY
                                           ,MEM_RD_ZY])

 [IOA, X, 0, 0, 1, 1, 1, 0]->[0, 0, 1, Z, 0, 0];" 1 IO read 16-Bit
 [IOA, X, 1, 0, 1, 1, 1, 0]->[Z, 0, 1, Z, 0, 0];" 2 IO read  8-Bit
 [IOA, X, 0, 1, 0, 1, 1, 0]->[0, 1, 0, Z, 0, 0];" 3 IO write 16-Bit
 [IOA, X, 1, 1, 0, 1, 1, 0]->[Z, 1, 0, Z, 0, 0];" 4 IO write  8-Bit
 [  X,MA, 0, 1, 1, 0, 1, 0]->[Z, 0, 0, 0, 0, 1];" 5 MEM read  16-Bit
 [  X,MA, 0, 1, 1, 1, 0, 0]->[Z, 0, 0, 0, 1, 0];" 6 MEM write 16-Bit
 [  X, X, 1, 1, 1, 1, 1, 1]->[Z, 0, 0, Z, 0, 0];" 7 kein Zugriff
 [  X, X, 1, 1, 1, 1, 1, 1]->[Z, 0, 0, Z, 0, 0];" 8 kein Zugriff
 [  X, X, 1, 1, 1, 1, 1, 1]->[Z, 0, 0, Z, 0, 0];" 9 kein Zugriff
 [  X, X, 1, 1, 1, 1, 1, 1]->[Z, 0, 0, Z, 0, 0];"10 kein Zugriff

 test_vectors
"------------------------------------------------------------------
" berpr fen der beiden Zustandsmaschinen
"Zun chst Register lesen
([INTAKT,EXTAKT,RESET,SBHE~,IOADRES,IOR~,IOW~,MADRES,MEMR~,MEMW~,FREE~,AEN]
->TEST_ZYK)
"            I                              D A   D     A
"   I E      O        M           Z   D E D D I W R   D
"   N X R    A        A           E   E X R I N R D R R
"   T T E S  D        D  M M F    R W X _ _ N _ _ _ E _
"   A A S B  R  I I   R  E E R A  O A _ L L _ L I I A I
"   K K E H  E  O O   E  M M E E  W I E A A E A N N D N
"   T T T E  S  R W   S  R W E N  S T N T T N T T T Y C
"         ~     ~ ~      ~ ~ ~    ~ ~ ~     ~   ~ ~
 [  C,C,1,X,  X ,X,X,  X ,X,X,1,X]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"11 Reset-Zustand
 [  C,C,0,1,  X ,1,1,  X ,1,1,1,X]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"12 Beide Maschinen frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"13

"Simulation: Einfaches Register lesen
 [ C,C,0,0,IOA ,0,1,  X ,1,1,1,0]-> [ 0,1,0,0,0,1,1,Z,Z,0,1];"14 Lesezugriff auf Regis-
ter
 [ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"15 Warten bis intern frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,0,1,0,1,0,0,1];"16 Internes Lesen
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,0,1,1,1,0,1,0];"17 Datum intern latchen
```

```
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"18 Beide Maschinen frei
"Simulation: Register lesen (2 Takte), dann erneut lesen mit warten
"           I                              D A  D    A
"  I E      O         M             Z  D E D D I W R  D
"  N X R    A         A             E  E X R I N R D R R
"  T T E S  D         D  M M F      R W X _ _  N _ _ _  E _
"  A A S B  R  I I    R  E E R A    O A _ L L _ L I I A I
"  K K E H  E  O O    E  M M E E    W I E A A E A N N D N
"  T T T E  S  R W    S  R W E N    S T N T T N T T T Y C
"        ~      ~ ~       ~ ~ ~      ~ ~ ~     ~    ~ ~
[ C,C,0,0,IOA ,0,1,  X ,1,1,1,0]-> [ 0,1,0,0,0,1,1,Z,Z,0,1];"19 Lesezugriff auf Regis-
ter
[ C,C,0,0,IOA ,0,1,  X ,1,1,1,0]-> [ 0,1,0,0,0,1,1,Z,Z,0,1];"20 dauert etwas l nger
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"21 Warten bis intern frei
[ C,C,0,1,IOA ,0,1,  X ,1,1,0,0]-> [ 1,0,1,0,0,1,0,1,0,0,1];"22 Internes Lesen, Bus
will schon wieder
[ C,C,0,1,IOA ,0,1,  X ,1,1,0,0]-> [ 1,0,1,0,0,1,1,1,0,1,0];"23 Datum intern latchen
[ C,C,0,0,IOA ,0,1,  X ,1,1,1,0]-> [ 0,1,1,0,0,1,1,Z,Z,0,1];"24 Bus muss noch warten
[ C,C,0,1,IOA ,0,1,  X ,1,1,1,0]-> [ 0,1,0,0,0,1,1,Z,Z,0,1];"25 Warten bis intern frei
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,0,1,1,1,1,0,1];"26 warten auf internes
Lesen
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,0,1,0,1,0,0,1];"27 Internes Lesen
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,0,1,1,1,0,1,0];"28 Datum intern latchen
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"29 Beide Maschinen frei
"-----------------------------------------------------------------------------
"Jetzt Register schreiben
"           I                              D A  D    A
"  I E      O         M             Z  D E D D I W R  D
"  N X R    A         A             E  E X R I N R D R R
"  T T E S  D         D  M M F A    R W X _ _  N _ _ _  E _
"  A A S B  R  I I    R  E E R E    O A _ L L _ L I I A I
"  K K E H  E  O O    E  M M E N    W I E A A E A N N D N
"  T T T E  S  R W    S  R W E      S T N T T N T T T Y C
"        ~      ~ ~       ~ ~ ~      ~ ~ ~     ~    ~ ~
[ C,C,1,X,  X ,X,X,  X ,X,X,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"30 Reset-Zustand
[ C,C,1,X,  X ,X,X,  X ,X,X,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"31 Reset-Zustand
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"32 Beide Maschinen frei
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"33
"Simulation: Einfaches Register schreiben
[ C,C,0,0,IOA ,1,0,  X ,1,1,1,0]-> [ 0,1,1,1,0,1,1,Z,Z,0,1];"34 Schreibzugriff auf
Register
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,1,0,1,1,Z,Z,0,1];"35 Warten bis intern frei
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,0,0,1,1,1,0,1];"36
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,0,0,1,0,1,1,0];"37 Internes Schreiben
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"38 Beide Maschinen frei
"Simulation: Register schreiben (2 Takte), dann erneut schreiben mit warten
"           I                              D A  D    A
"  I E      O         M             Z  D E D D I W R  D
"  N X R    A         A             E  E X R I N R D R R
"  T T E S  D         D  M M F A    R W X _ _  N _ _ _  E _
"  A A S B  R  I I    R  E E R E    O A _ L L _ L I I A I
"  K K E H  E  O O    E  M M E N    W I E A A E A N N D N
"  T T T E  S  R W    S  R W E      S T N T T N T T T Y C
"        ~      ~ ~       ~ ~ ~      ~ ~ ~     ~    ~ ~
[ C,C,0,0,IOA ,1,0,  X ,1,1,1,0]-> [ 0,1,1,1,0,1,1,Z,Z,0,1];"39 Schreibzugriff auf
Register
[ C,C,0,0,IOA ,1,0,  X ,1,1,1,0]-> [ 0,1,1,1,0,1,1,Z,Z,0,1];"40 dauert etwas l nger
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,1,0,1,1,Z,Z,0,1];"41 Warten bis intern frei
```

```
[ C,C,0,1,IOA ,1,0,  X ,1,1,0,0]-> [ 1,0,1,1,0,0,1,1,1,0,1];"42 Bus will schon wieder
[ C,C,0,1,IOA ,1,0,  X ,1,1,0,0]-> [ 1,0,1,1,0,0,1,0,1,1,0];"43
[ C,C,0,0,IOA ,1,0,  X ,1,1,1,0]-> [ 0,1,1,0,0,1,1,Z,Z,0,1];"44 Bus muss noch warten
[ C,C,0,1,IOA ,1,0,  X ,1,1,1,0]-> [ 0,1,1,1,0,1,1,Z,Z,0,1];"45 Warten bis intern frei
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,0,1,1,1,1,0,1];"46 warten auf internes
schreiben
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,0,0,1,1,1,0,1];"47
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,0,0,1,0,1,1,0];"48 Internes schreiben
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"49 Beide Maschinen frei

test_vectors
"Zun chst Speicher lesen
([INTAKT,EXTAKT,RESET,SBHE~,IOADRES,IOR~,IOW~,MADRES,MEMR~,MEMW~,FREE~,AEN]
->TEST_ZYK)
"          I                        D A  D   A
"   I E    O       M          Z  D E D D I W R  D
"   N X R  A       A          E  E X R I N R D R R
"   T T E S D      D M M F    R W X _ _ N _ _ _ E _
"   A A S B R I I  R E E R A  O A _ L L _ L I I A I
"   K K E H E O O  E M M E E  W I E A A E A N N D N
"   T T T E S R W  S R W E N  S T N T T N T T T Y C
"         ~   ~ ~    ~ ~ ~    ~ ~ ~     ~   ~ ~
[ C,C,1,X,  X ,X,X,  X ,X,X,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"50 Reset-Zustand
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"51 Beide Maschinen frei
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"52
"Simulation: Einfaches Speicher lesen
[  C,C,0,0,  X ,1,1, MA ,0,1,1,0]-> [ 0,1,0,0,1,1,1,Z,Z,0,1];"53 Lesezugriff auf
Speicher
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,1,1,1,Z,Z,0,1];"54 Warten bis intern frei
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,1,1,0,1,0,0,1];"55 Internes Lesen
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,1,1,1,1,0,1,0];"56 Datum intern latchen
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"57 Beide Maschinen frei
"Simulation: Speicher lesen (2 Takte), dann erneut lesen mit warten
"          I                        D A  D   A
"   I E    O       M          Z  D E D D I W R  D
"   N X R  A       A          E  E X R I N R D R R
"   T T E S D      D M M F    R W X _ _ N _ _ _ E _
"   A A S B R I I  R E E R A  O A _ L L _ L I I A I
"   K K E H E O O  E M M E E  W I E A A E A N N D N
"   T T T E S R W  S R W E N  S T N T T N T T T Y C
"         ~   ~ ~    ~ ~ ~    ~ ~ ~     ~   ~ ~
[  C,C,0,0,  X ,1,1, MA ,0,1,1,0]-> [ 0,1,0,0,1,1,1,Z,Z,0,1];"58 Lesezugriff auf
Speicher
[ C,C,0,0,  X ,1,1, MA ,0,1,1,0]-> [ 0,1,0,0,1,1,1,Z,Z,0,1];"59 dauert etwas l nger
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,1,1,1,Z,Z,0,1];"60Warten bis intern frei
[ C,C,0,1,  X ,1,1, MA ,0,1,0,0]-> [ 1,0,1,0,1,1,0,1,0,0,1];"61 Internes Lesen, Bus
will schon wieder
[ C,C,0,1,  X ,1,1, MA ,0,1,0,0]-> [ 1,0,1,0,1,1,1,1,0,1,0];"62 Datum intern latchen
[ C,C,0,0,  X ,1,1, MA ,0,1,1,0]-> [ 0,1,1,0,0,1,1,Z,Z,0,1];"63 Bus muss noch warten
[ C,C,0,1,  X ,1,1, MA ,0,1,1,0]-> [ 0,1,0,0,1,1,1,Z,Z,0,1];"64 Warten bis intern frei
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,1,1,1,1,1,0,1];"65 warten auf internes
Lesen
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,1,1,0,1,0,0,1];"66 Internes Lesen
[ C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,0,1,1,1,1,0,1,0];"67 Datum intern latchen
[ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"68 Beide Maschinen frei
"---------------------------------------------------------------------- ----
"Jetzt Speicher schreiben
"          I                        D A  D   A
"   I E    O       M          Z  D E D D I W R  D
```

```
"    N X R      A         A                E  E X R I N R D R R
"    T T E S    D         D  M M F         R W X     N       E
"    A A S B    R  I I    R  E E R A       O A _ L L _ L I I A I
"    K K E H    E  O O    E  M M E E       W I E A A E A N N D N
"    T T T E    S  R W    S  R W E N       S T N T T N T T T Y C
"          ~       ~ ~       ~ ~ ~         ~ ~ ~     ~     ~ ~
 [  C,C,1,X,  X ,X,X,  X ,X,X,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"69 Reset-Zustand
 [  C,C,1,X,  X ,X,X,  X ,X,X,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"70 Reset-Zustand
 [ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"71 Beide Maschinen frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"72
"Simulation: Einfaches Speicher schreiben
 [  C,C,0,0,  X ,1,1, MA ,1,0,1,0]-> [ 0,1,1,1,1,1,1,Z,Z,0,1];"73 Schreibzugriff auf
Speicher
 [ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,1,1,1,1,Z,Z,0,1];"74 Warten bis intern frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,1,0,1,1,1,0,1];"75
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,1,0,1,0,1,1,0];"76 Internes Schreiben
 [ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"77 Beide Maschinen frei
"Simulation: Speicher schreiben (2 Takte), dann erneut schreiben mit warten
"               I                            D A  D         A
"    I E        O         M             Z  D E D D I W R  D
"    N X R      A         A             E  E X R I N R D R R
"    T T E S    D         D  M M F      R W X _ _ N _ _ _ E _
"    A A S B    R  I I    R  E E R A    O A _ L L _ L I I A I
"    K K E H    E  O O    E  M M E E    W I E A A E A N N D N
"    T T T E    S  R W    S  R W E N    S T N T T N T T T Y C
"          ~       ~ ~       ~ ~ ~      ~ ~ ~     ~     ~ ~
 [  C,C,0,0,  X ,1,1, MA ,1,0,1,0]-> [ 0,1,1,1,1,1,1,Z,Z,0,1];"78 Schreibzugriff auf
Speicher
 [  C,C,0,0,  X ,1,1, MA ,1,0,1,0]-> [ 0,1,1,1,1,1,1,Z,Z,0,1];"79 dauert etwas l nger
 [ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,1,1,1,1,Z,Z,0,1];"80 Warten bis intern frei
 [ C,C,0,1,  X ,1,1, MA ,1,0,0,0]-> [ 1,0,1,1,1,0,1,1,1,0,1];"81 Bus will schon wieder
 [  C,C,0,1,  X ,1,1, MA ,1,0,0,0]-> [ 1,0,1,1,1,0,1,0,1,1,0];"82
 [ C,C,0,0,  X ,1,1, MA ,1,0,1,0]-> [ 0,1,1,0,0,1,1,Z,Z,0,1];"83 Bus muss noch warten
 [ C,C,0,1,  X ,1,1, MA ,1,0,1,0]-> [ 0,1,1,1,1,1,1,Z,Z,0,1];"84 Warten bis intern frei
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,1,1,1,1,1,0,1];"85 warten auf internes
schreiben
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,1,0,1,1,1,0,1];"86
 [  C,C,0,1,  X ,1,1,  X ,1,1,0,0]-> [ Z,Z,1,1,1,0,1,0,1,1,0];"87 Internes schreiben
 [ C,C,0,1,  X ,1,1,  X ,1,1,1,0]-> [ Z,Z,1,0,0,1,1,Z,Z,0,1];"88 Beide Maschinen frei

end
```

FITTER FILE: MACH210

Note: This File has been condensed in order to save trees.

```
AMD MACH FITR - MARKET RELEASE (1-24-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990

Flags Used:        Unplace=True              Max Packing=True
Flags Used:      Expand Small=False         Expand All=False

************************************************************
     Mach PLD Fitter - v 1.46           ISA-BUS-INTERFACE
************************************************************

*** Timing Analysis for Signals
 Parameter   Min   Max          Signal List (Those having Max delay.)
      Tpd      1     1          IOCS16~        MEMCS16~
      Tsu      1     2          ZEROWS~          WAIT~          DEX_EN~
                                DEX_LAT        ADR_LAT
      Tco      0     0            READY          WAIT~          DEX_LAT
                                ZEROWS~        DEX_EN~          ADR_LAT
                                DIN_EN~        ADR_INC
      Tcr      1     1          ZEROWS~          WAIT~          DEX_LAT
                                ADR_LAT        DIN_EN~          DEX_EN~
                                DIN_LAT          READY

 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Register clock to combinatorial output
 Tcr - Register thru combinatorial logic to setup
 All delay values are expressed in terms of array passes

*** Device Resource Checks
                Available         Used         Remaining
      Clocks:       2               2             0
        Pins:      38              37             1     ->     97%
   I/O Macro:      32              13            19
 Total Macro:      64              18            46
 Product Terms:   256              72           144     ->     42%

MACH-PLD Resource Checks OK!

Partitioning Design into Blocks...

*** Last Equations Placed in Blocks

Weakly -            SEL~
Assign -

*** Block Partitioning Results

                Array    Macros     # I/O     Buried   Product     Signal
               Inputs    Remain     Macro     Logic     Terms      Fanout
   Block-> A     12        10         5         1        48          8
   Block-> B     15        13         1         2        12          3
   Block-> C     10        10         6         0        40          7
   Block-> D      9        13         1         2        12          3

*** Block Signal List
```

```
Block-> A          SEL~          ADR_LAT          DEX_EN~          DEX_LAT
                   WAIT~         ZEROWS~

Block-> B       IO_RD_ZY        IO_WR_ZY       IOCS16~_I          IOCS16~

Block-> C        READY_I           READY         ADR_INC          WR_INT~
                 RD_INT~         DIN_LAT         DIN_EN~

Block-> D      MEM_RD_ZY       MEM_WR_ZY      MEMCS16~_I         MEMCS16~

|> INFORMATION F050 - Device Utilization....... *: 64 %
```

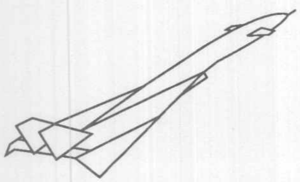
```
*** Feedback Map - ISA-BUS-INTERFACE

 Gbl Inp .--.      I/O   .--+--A--+--.  I/O             I/O  .--+--B--+--.  I/O
         | 0|  /WAIT~ : 0|     |21| IO_WR_Z               | 0|     |21|
         | 1|  MEM_WR_Z  1|     |20| IO_RD_Z          A6 : 1|     |20| A9
         | 2|  ADR_LAT : 2|     |19|               SBHE~ : 2|     |19|
         | 3|     SEL~ : 3|     |18|                 AEN : 3|     |18| A8
         | 4|  DEX_LAT : 4|     |17|             /IOCS16~  4|     |17| A7
         | 5|  /DEX_EN~  5|     |16|                  A0 : 5|     |16|
         '--'   /READY : 6|     |15|                  A1 : 6|     |15|
               ZEROWS~ : 7|     |14|                  A2 : 7|     |14|
                 RESET : 8|     |13|                  A3 : 8|     |13: IOW~
                         | 9|     |12: MEM_RD_        A4 : 9|     |12: IOR~
                         |10|     |11|                A5 :10|     |11|
                         '--+--C--+--'                  '--+--D--+--'
                /WAIT~ : 0|     |21|                 A21 : 0|     |21|
                         | 1|     |20|               A22 : 1|     |20|
               DIN_EN~ : 2|     |19|               SBHE~ : 2|     |19|
                         | 3|     |18|               AEN : 3|     |18|
               DEX_LAT : 4|     |17|               MEMR~ : 4|     |17|
               /READY_I  5|     |16| RESET         MEMW~ : 5|     |16|
               RD_INT~ : 6|     |15|                     | 6|     |15|
               ADR_INC : 7|     |14|                     | 7|     |14|
               WR_INT~ : 8|     |13|             /MEMCS16  8|     |13|
                         | 9|     |12: DIN_LAT             | 9|     |12|
                         |10|     |11: FREE~          A20 :10|     |11: A23
                         '--+-u--u+--'                  '--+-u--u+--'

*** Logic Map - ISA-BUS-INTERFACE

 Gbl Inp .--.      I/O   .--+--A--+--.  I/O             I/O  .--+--B--+--.  I/O
   SBHE~| 0|    /WAIT~ | 0|10    |21|             /IOCS16~  0| 1    |21|
   RESET| 1|           | 1| *    |20|             IO_RD_ZY  1| 1    |20|
  INTAKT| 2|           | 2| *    |19|             IO_WR_ZY  2| 1    |19|
     AEN| 3|           | 3| *    |18|                     | 3| .    |18|
   FREE~| 4|  /DEX_EN~  4| 9    |17|                     | 4| .    |17|
  EXTAKT| 5|           | 5| *    |16|                     | 5| .    |16|
        '--'  ZEROWS~ | 6| 5   .|15|                 A1 | 6| .   .|15|
                      | 7| *   .|14|                    | 7| .   .|14|
              ADR_LAT | 8| 5   .|13|                    | 8| .   .|13|
                 SEL~ | 9| 1   .|12|                    | 9| .   .|12|
              DEX_LAT |10| 4   *|11|                    |10| .   .|11|
                      '--+--C--+--'                     '--+--D--+--'
               /READY | 0| 6    |21|            /MEMCS16  0| 1    |21|
                      | 1| *    |20|                    | 1| .    |20|
              RD_INT~ | 2| 6    |19|            MEM_WR_Z  2| 1    |19|
                      | 3| *    |18|                    | 3| .    |18|
              DIN_LAT | 4| 7    |17|                    | 4| .    |17|
                      | 5| *    |16|                    | 5| .    |16|
              DIN_EN~ | 6| 6   .|15|                    | 6| .   .|15|
                      | 7| *   .|14|                    | 7| .  1|14| MEM_RD_
              WR_INT~ | 8| 4   .|13|                    | 8| .   .|13|
                      | 9| .   .|12|                    | 9| .   .|12|
              ADR_INC |10| 3   .|11|                    |10| .   .|11|
                      '--+-u--u+--'                     '--+-u--u+--'
```

```
*** Pin Map - ISA-BUS-INTERFACE

                                 /WAIT~
                                  A6  |        /MEMCS16~
                         /DEX_EN~  |  |        |  A20
                       ZEROWS~  |  |  |        |  |  A21
                    ADR_LAT  |  |  |  |        |  |  |  A22
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
             DEX_LAT| 7                                   39|A23
                  A7| 8                  G  V             38|IOR~
                  A8| 9                  n  c             37|IOW~
               SBHE~|10                  d  c             36|A0
               RESET|11                                   35|EXTAKT
                Gnd |12            MACH-210               34| Gnd
              INTAKT|13                                   33|FREE~
                  A5|14             V  G                  32|AEN
                  A4|15             c  n                  31|
                  A3|16             c  d                  30|A9
                  A2|17                                   29|ADR_INC
                    |   1  1  2  2  2  2  2  2  2  2  2     |
                    |   8  9  0  1  2  3  4  5  6  7  8     |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                       A1  |  |  |        |  |  |  |  WR_INT~
                       MEMW~  |  |        |  |  |  DIN_EN~
                          MEMR~  |        |  |  DIN_LAT
                          /IOCS16~        |  RD_INT~
                                          /READY

The Design Doc is  stored in ===> isa.Rpt
The Jedec Data is  stored in ===> isa.Jed
The Placements are stored in ===> isa.Plc
The  Fuse Plot is  stored in ===> isa.Xpt

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: isa
```

PLACEMENT FILE: MACH210

```
; Flags Used:          Unplace=True               Max Packing=True
; Flags Used:     Expand Small=False              Expand All=False

; FITR generated placements

Pin   35         EXTAKT  ; Inp        ; I  5
Pin   13         INTAKT  ; Inp        ; I  2
Pin   10          SBHE~  ; Inp        ; I  0
Pin   20          MEMR~  ; Inp        ; B  2
Pin   19          MEMW~  ; Inp        ; B  4
Pin   38           IOR~  ; Inp        ; D 10
Pin   37           IOW~  ; Inp        ; D 12
Pin   33          FREE~  ; Inp        ; I  4
Pin   11          RESET  ; Inp        ; I  1
Pin   32            AEN  ; Inp        ; I  3
Pin   39            A23  ; Inp        ; D  8
Pin   40            A22  ; Inp        ; D  6
Pin   41            A21  ; Inp        ; D  4
Pin   42            A20  ; Inp        ; D  2
Pin   30             A9  ; Inp        ; C 12
Pin    9             A8  ; Inp        ; A 14
Pin    8             A7  ; Inp        ; A 12
Pin    3             A6  ; Inp        ; A  2
Pin   14             A5  ; Inp        ; B 14
Pin   15             A4  ; Inp        ; B 12
Pin   16             A3  ; Inp        ; B 10
Pin   17             A2  ; Inp        ; B  8
Pin   18             A1  ; Inp        ; B  6
Pin   36             A0  ; Inp        ; D 14
Pin   21      /IOCS16~  Comb          ; B  0
Pin   43     /MEMCS16~  Comb          ; D  0
Node   8     ZEROWS~_I   Reg          ; A  6
Node   2      /WAIT~_I   Reg          ; A  0
Node  40     DIN_EN~_I   Reg          ; C  6
Node   6    /DEX_EN~_I   Reg          ; A  4
Node  38     DIN_LAT_I   Reg          ; C  4
Node  12     DEX_LAT_I   Reg          ; A 10
Node  10     ADR_LAT_I   Reg          ; A  8
Node  44     ADR_INC_I   Reg          ; C 10
Node  42     WR_INT~_I   Reg          ; C  8
Node  36     RD_INT~_I   Reg          ; C  2
Pin   24        /READY   Reg          ; C  0
Node  52     MEM_WR_ZY  Comb          ; D  2
Node  64     MEM_RD_ZY  Comb          ; D 14
Node  20      IO_WR_ZY  Comb          ; B  2
Node  19      IO_RD_ZY  Comb          ; B  1
Node  11          SEL~  Comb          ; A  9
Pin    2        /WAIT~   Reg          ; A  0
Pin    7       DEX_LAT   Reg          ; A 10
Pin    5       ZEROWS~   Reg          ; A  6
Pin    4      /DEX_EN~   Reg          ; A  4
Pin    6       ADR_LAT   Reg          ; A  8
Pin   27       DIN_EN~   Reg          ; C  6
Pin   26       DIN_LAT   Reg          ; C  4
Pin   28       WR_INT~   Reg          ; C  8
Pin   25       RD_INT~   Reg          ; C  2
Node  34     /READY_I    Reg          ; C  0
```

```
Pin   29          ADR_INC    Reg         ; C 10
Node  18       /IOCS16~_I   Comb        ; B  0
Node  50      /MEMCS16~_I   Comb        ; D  0

; Group Mach_Seg_A SEL~ ADR_LAT DEX_EN~ DEX_LAT WAIT~ ZEROWS~
; Group Mach_Seg_B IO_RD_ZY IO_WR_ZY IOCS16~_I IOCS16~
; Group Mach_Seg_C READY_I READY ADR_INC WR_INT~ RD_INT~ DIN_LAT DIN_EN~
; Group Mach_Seg_D MEM_RD_ZY MEM_WR_ZY MEMCS16~_I MEMCS16~
```



Analog Acquisition Interface: MACH210

Advanced Micro Devices

Application Note

by Stephen Leis, Northern Digital Inc., Canada

OVERVIEW

The following design utilizes a MACH210 as a controller for an existing analog acquisition unit. The application required interfacing the 16 channel, 12-bit remote unit to a micro-controller. The acquisition unit has three signals as input and three signals as output. The block diagram is shown in Figure 7-1.

The first input signal is called frame_strobe. When this signal is activated the acquisition unit will start converting the analog inputs sequentially starting with channel 0 at a rate of one per 32 fs. This will continue until frame_strobe is deactivated. The second and third inputs are used as digital control of a programmable gain amplifier, and are labelled gn1, and gn2.

The acquisition unit has a two wire serial interface. Each channel produces 12 data bits which are valid on the rising edge of the serial clock. A third output is used to signal end of data conversion on a channel basis. The output are labelled serial_data_in, and serial_clock_in. The timing diagram is shown in Figure 7-2.

MACH210 CONTROLLER

A single MACH210 plus a 9-bit by 512-deep FIFO chip will interface to the acquisition unit. The twelve data bits are stored in two consecutive locations in the FIFO in the following format.

Table 7-1. Data Bit Storage Format

Bit	9	8	7	6	5	4	3	2	1	0
First Word	1	Channel Number				4 MSB of Data				
Second Word	0	8 LSB of Data								

At maximum sampling rate the FIFO must be drained by the micro-controller every 8 ms.

The controller has a 6-bit command buffer, 12-bit shift register, multiplexer, 4-bit channel counter and an 8-state ASM (Algorithmic State Machine). The micro-controller will write the number of channels to be converted in a frame to the command buffer. The top two bits will be used for the gain. The micro-controller then issues a frame pulse once per frame and the ASM does the rest. Data is written to the FIFO twice per channel.

The state machine controller will start in state 0, in which the channel counter is continuously being cleared. The start of frame will be sent to the acquisition unit on command of the micro-controller. This will also allow the transition to state 1. The state diagram is shown in Figure 7-3.

In state 1 the ASM will idle waiting for the END of CONVERSION line to become inactive. This signals the conversion is starting. The channel counter is compared to the number of channels in the command register. If they are equal the frame strobe is deactivated. This allows the code 0 to represent a single channel.

The ASM will idle in state 2 waiting for the END of CONVERSION line to become active once more. During this state the acquisition unit is sending data in serial to the internal 12-bit shift register.

In state 3 the FIFO write command is set up, to be activated in the next state.

The upper four bits of data are enable to the FIFO lower inputs during state 4. The channel number is placed in the next higher four bits, and the ninth bit (TAG BIT) is set high. The write pulse takes place during this state.

The write pulse is now inactive during state 5, and is set up to be activated again in state 6. The data is held to the inputs of the FIFO.

The lower 8 bits of data is connected to the input of the FIFO during state 6. The tag bit is set to zero. The write pulse is active now. The channel counter is incremented at the end of this state.

The data is held on the inputs of the FIFO during state 7. The write pulse is complete. At the end of the state a decision is made. If the frame strobe is still active then not all the channels have been converted. In this case control is given to state 1. If the frame strobe is inactive the frame is complete and control is given to state 0.

Publication# 17756 Rev. A Amendment /0
Issue Date: May 1993

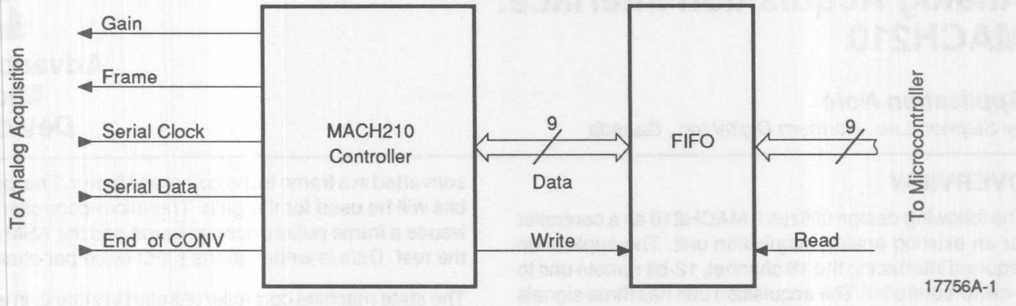


Figure 7-1. Block Diagram of Analog Acquisition Interface

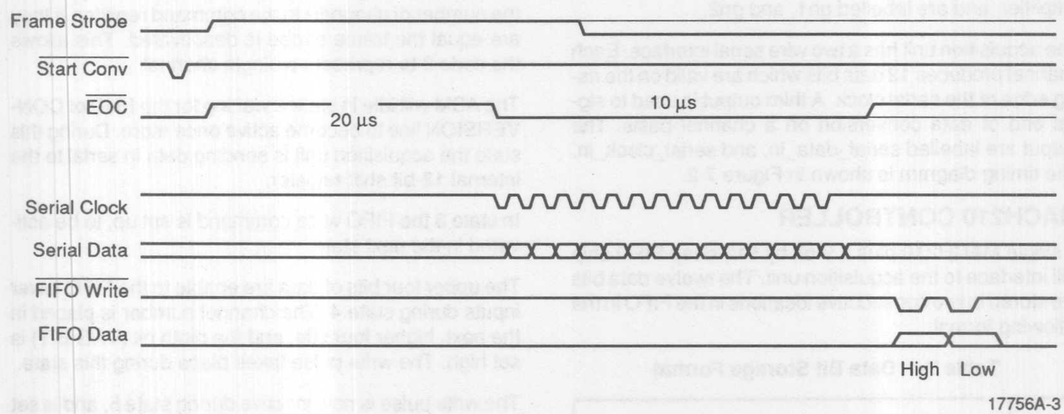


Figure 7-2. Timing Diagram of Analog Conversion

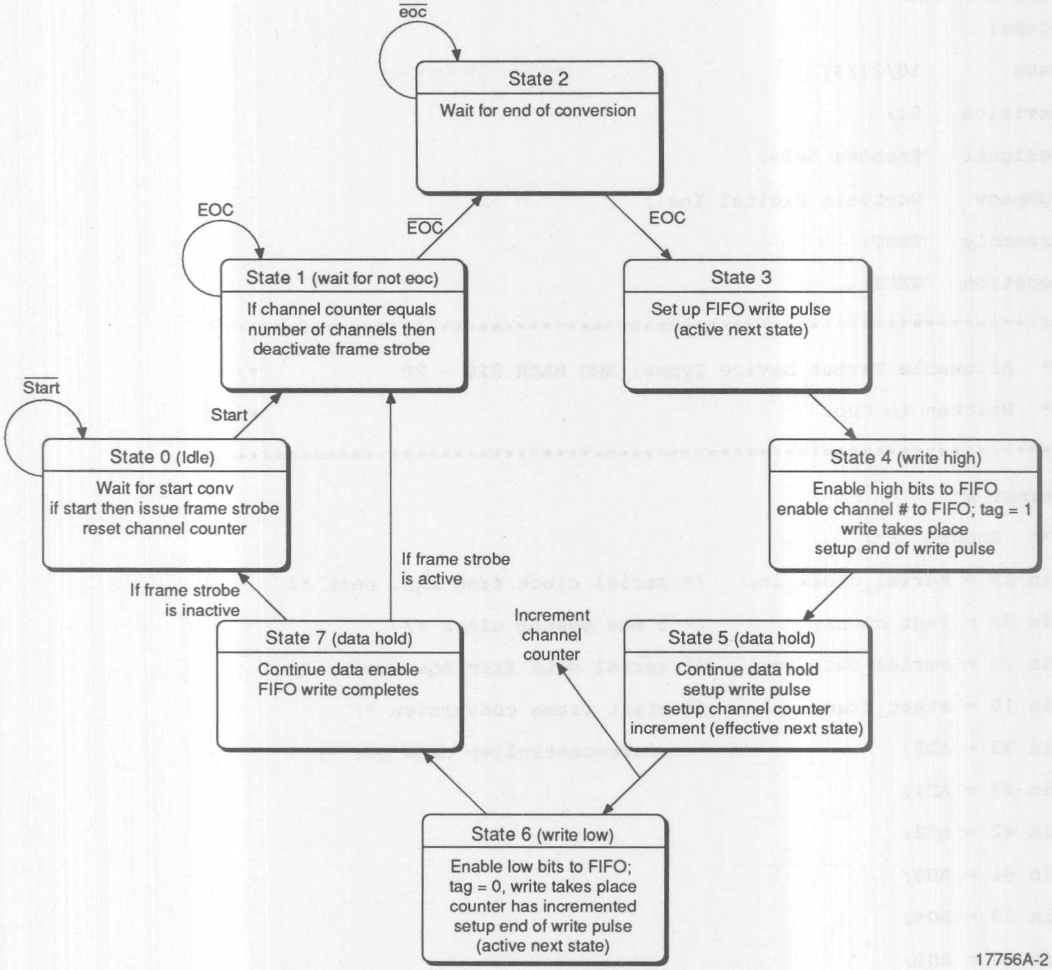


Figure 7-3. State Diagram of Analog Acquisition Interface

DESIGN FILE

```
scope;

Date        10/21/91;

Revision    01;

Designer    Stephen Leis;

Company     Northern Digital Inc.;

Assembly    TEST;

Location    TEST;

/******************************************************************/

/*  Allowable Target Device Types: AMD MACH 210 - 20            */

/*  Written in Cupl                                              */

/******************************************************************/

format d;

/**  Inputs  **/

Pin 13 = serial_clock_in;   /* serial clock from Aqu. unit */

Pin 35 = fast_clock;        /* 5 MHz system clock */

Pin 36 = serial_data_in;    /* serial data from Aqu. unit   */

Pin 10 = start_frame;       /* start frame conversion */

Pin 33 = AD0;               /* micro-controller data bus */

Pin 43 = AD1;

Pin 42 = AD2;

Pin 41 = AD3;

Pin 39 = AD4;

Pin 38 = AD5;

Pin 32 = notEOC;            /* not end of conversion from Aqu. unit */

Pin 11 = not_write_command; /* write command to this device */

Pin 37 = reset;             /* master reset pin */

EOC         = !notEOC;

write_command = !not_write_command;

/** outputs **/

Pin  6 = frame_strobe;      /* frame strobe to Aqu. unit */

Pin 24 = gn1;              /* gain code 1 to Aqu. unit */

Pin 25 = gn2;              /* gain code 2 to Aqu. unit */

Pin  7 = !fifo_write;      /* write fifo command  */

Pin 20 = fifo0;            /* fifo input data */
```

```
Pin 19 = fifo1;
Pin 18 = fifo2;
Pin 17 = fifo3;
Pin  3 = fifo4;
Pin  4 = fifo5;
Pin  5 = fifo6;
Pin  2 = fifo7;
Pin 21 = fifo8;
/** buried **/
Pinnode 76 = ZZ11;    /* internal 12 bit shift register */
Pinnode 71 = ZZ10;
Pinnode 75 = ZZ9;
Pinnode 92 = ZZ8;
Pinnode 94 = ZZ7;
Pinnode 85 = ZZ6;
Pinnode 93 = ZZ5;
Pinnode 83 = ZZ4;
Pinnode 68 = ZZ3;
Pinnode 64 = ZZ2;
Pinnode 62 = ZZ1;
Pinnode 84 = ZZ0;
Pinnode 82 = ch0;     /* number of channels to be converted */
Pinnode 86 = ch1;
Pinnode 80 = ch2;
Pinnode 81 = ch3;
Pinnode 50 = ct0;     /* number of channels that have been
converted */
Pinnode 54 = ct1;
Pinnode 56 = ct2;
Pinnode 58 = ct3;
Pinnode 46 = ctl0;    /* state vector for state machine */
Pinnode 59 = ctl1;
Pinnode 60 = ctl2;
Pinnode 107 = EL;     /* used for 4 bit equality test */
Pinnode 96 = EH;
```

```
Pinnode 48 = start; /* start conversion frame synchronized to
clock */
Pinnode 57 = EOC_qual;  /* EOC synchronized to clock */
/***********************************************************/
/* async latch to buffer data                               */
/*                                                          */
/*  user enters number of channels in frame - 1 (0..15)     */
/*  and 2 bit gain code (0..3)                              */
/*  d0..d5 => ch0..ch3,gn1,gn2                              */
/*                                                          */
/***********************************************************/
min ch0 = 0;
min ch1 = 0;
min ch2 = 0;
min ch3 = 0;
min gn1 = 0;
min gn2 = 0;
field channels = [ch0..ch3];
LE = !write_command;
/* the following equations represent a transparent latch  and */
/* were derived using asynchronous sequential logic techniques.*/
/* The use of these equations effectively increases the number */
/* of clock inputs into the mach device                        */
$repeat i=[0..3]
    ch{i} = ch{i} & LE  #  AD{i} & ch{i}  #  AD{i} & !LE;
$repend
gn1 = gn1 & LE  #  AD4 & gn1  #  AD4 & !LE;
gn2 = gn2 & LE  #  AD5 & gn2  #  AD5 & !LE;
/***********************************************************/
/* start frame conversion sequence (async to sync)          */
/*                                                          */
/*  This flag is set by asynchronous signal start frame and */
/*  is reset when the conversion process is started.        */
/*                                                          */
/***********************************************************/
```

```
start.ckmux = fast_clock;
start.ar = reset;
start.t =  start_frame & !start
    # start & ctl1;
/*****************************************************************/
/*                                                              */
/* control state machine                                        */
/*                                                              */
/* this 8 state algorithmic state machine sequences the device*/
/* through its operations.                                      */
/*                                                              */
/*  Idle in state 0 until start                                 */
/*          - clear channel counter                             */
/*          - issue frame strobe when start activated           */
/*                  (active next clock edge )                   */
/*                                                              */
/*  Idle in state 1 until conversion starts                     */
/*          - compare channel counter with user channels        */
/*                - if equal deactivate frame_strobe            */
/*                  (active next clock edge )                   */
/*                                                              */
/*  Idle in state 2 until conversion is complete                */
/*                                                              */
/*  State 3                                                     */
/*          - issue write fifo command (active in state 4)      */
/*                                                              */
/*  State 4                                                     */
/*          - enable high bits of data to fifo                  */
/*          - enable channel number to fifo                     */
/*          - fifo write takes place                            */
/*                                                              */
/*  State 5                                                     */
/*          - continue enable high bits of data to fifo         */
/*          - continue enable channel number to fifo            */
/*          - fifo write completes                              */
```

```
/*          - increment channel counter (active next state)     */
/*          - issue write fifo command (active next state)      */
/*                                                               */
/*  State 6                                                      */
/*          - enable low bits of data to fifo                    */
/*          - fifo write takes place                             */
/*          - channel counter increments                         */
/*                                                               */
/*  State 7                                                      */
/*          - continue enable low bits of data to fifo           */
/*          - fifo write completes                               */
/*              - if frame_strobe true go to state 1             */
/*              - if frame_strobe false go to state 0            */
/*                                                               */
/***************************************************************/
EOC_qual.ckmux = fast_clock;
EOC_qual.d = EOC;
EOC_qual.ar = reset;
field state=[ctl0..ctl2];
state.ckmux = fast_clock;
state.ar = reset;
sequence state
{
    Present 0
        if !start next 0;
        if start next 1;
    Present 1
        if EOC_qual next 1;
        if !EOC_qual next 2;
    Present 2
        if !EOC_qual next 2;
        if EOC_qual next 3;
    Present 3
        next 4;
    Present 4
```

```
        next 5;
    Present 5
        next 6;
    Present 6
        next 7;
    Present 7
        if frame_strobe next 1;
        if !frame_strobe next 0;
}
/*****************************************************************/
/*                                                               */
/* channel counter                                               */
/*                                                               */
/*this four bit binary counter holds the number of channels that*/
/* have been converted (offset by one). This counter is cleared */
/* at state 0 and is incremented once every state 5            */
/*                                                               */
/*****************************************************************/
field channel_counter = [ct0..ct3];
channel_counter.ckmux = fast_clock;
channel_counter.ar = reset;
        /* sync reset */    /* increment counter */
ct0.t = ct0 & (state:0)  #  (state:5);
ct1.t = ct1 & (state:0)  #  ct0 & (state:5);
ct2.t = ct2 & (state:0)  #  ct1 & ct0 & (state:5);
ct3.t = ct3 & (state:0)  #  [ct2..ct0]:& & (state:5);
/************************************************************/
/*                                                          */
/* output controls (synchronized by state machine)          */
/*                                                          */
/************************************************************/
frame_strobe.ckmux = fast_clock;
frame_strobe.ar = reset;
frame_strobe.d = start & (state:0)
               # frame_strobe & (state:[4..7])
```

```
                # frame_strobe & (state:[0..3]) & !done;
enable_low   = state:[6,7];
enable_high  = state:[4,5];
fifo_write.ckmux = fast_clock;
fifo_write.d = state:[3,5];
/************************************************************/
/*                                                          */
/* 4 bit comparator                                         */
/* done = true when channel_counter = channel               */
/*                                                          */
/************************************************************/
done = !EL & !EH;
EL = (ct0 $ ch0) # (ct1 $ ch1);
EH = (ct2 $ ch2) # (ct3 $ ch3);
/************************************************************/
/* 12 bit shift register (holds a single converted channel */
/*                                                          */
/*                                                          */
/************************************************************/
field shift_register = [ZZ0..11];
shift_register.ckmux = serial_clock_in;
ZZ0.d = serial_data_in;
ZZ1.d = ZZ0;
ZZ2.d = ZZ1;
ZZ3.d = ZZ2;
ZZ4.d = ZZ3;
ZZ5.d = ZZ4;
ZZ6.d = ZZ5;
ZZ7.d = ZZ6;
ZZ8.d = ZZ7;
ZZ9.d = ZZ8;
ZZ10.d = ZZ9;
ZZ11.d = ZZ10;
/************************************************************/
/*                                                          */
```

```
/* fifo data driver ( 16 line to 8 line multiplexer )      */
/*                                                          */
/*   selects 4 bit channel and top 4 bits of data          */
/*         or 8 lsb of data                                 */
/*   bit 9 of fifo is high for first case and low for      */
/*   second                                                 */
/*                                                          */
/**********************************************************/

fifo0 = enable_low & ZZ0  #  enable_high & ZZ8;
fifo1 = enable_low & ZZ1  #  enable_high & ZZ9;
fifo2 = enable_low & ZZ2  #  enable_high & ZZ10;
fifo3 = enable_low & ZZ3  #  enable_high & ZZ11;
fifo4 = enable_low & ZZ4  #  enable_high & ct0;
fifo5 = enable_low & ZZ5  #  enable_high & ct1;
fifo6 = enable_low & ZZ6  #  enable_high & ct2;
fifo7 = enable_low & ZZ7  #  enable_high & ct3;
fifo8 = enable_high;
```

FITTER FILE

Note: This file has been condensed in order to save trees.

```
AMD MACH FITR - BETA RELEASE (6-17-92)
(C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990,91,92

************************************************
     Mach PLD Fitter - v 1.67          scope
************************************************

*** Timing Analysis for Signals

 Parameter    Min    Max          Signal List (Those having Max delay.)
      Tpd      1      2                  EL                EH
      Tsu      1      3       FRAME_STROBE
      Tco      0      1               FIFO0             FIFO1             FIFO2
                                      FIFO3             FIFO4             FIFO5
                                      FIFO6             FIFO8
      Tcr      1      2       FRAME_STROBE
.

 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - Clock to output (register output to combinatorial output)
 Tcr -  Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
 All delay values are expressed in terms of array passes

*** Device Resource Checks

                 Available        Used         Remaining
         Clocks:     2              2              0
           Pins:    38             26             12      ->     68%
     I/O Macro:     32             13             19
   Total Macro:     64             40             24
 Product Terms:    256             83             92      ->     64%

MACH-PLD Resource Checks OK!

Partitioning Design into Blocks...

|> INFORMATION F050 - Device Utilization............. *: 64 %
```

```
                                      FIFO7
                                FIFO4 |        AD1
                             FIFO5 |  |        |  AD2
                          FIFO6 |  |  |        |  |  AD3
                FRAME_STROBE |  |  |  |        |  |  |  .
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
         /FIFO_WRITE| 7                                   39|AD4
                    | 8                  G  V             38|AD5
                    | 9                  n  c             37|RESET
         START_FRAME|1                   d  c             36|SERIAL_DATA_IN
     NOT_WRITE_COMMA|11                                   35|FAST_CLOCK
                 Gnd|12            MACH-210               34|Gnd
     SERIAL_CLOCK_IN|13                                   33|AD0
                    |14             V  G                  32|NOTEOC
                    |15             c  n                  31|
                    |16             c  d                  30|
               FIFO3|17                                   29|
                    |   1  1  2  2  2  2  2  2  2  2  2     |
                    |   8  9  0  1  2  3  4  5  6  7  8     |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                     FIFO2 |  |  |        |  |  |  |  '
                        FIFO1 |  |        |  |  |  '
                           FIFO0 |        |  |  '
                              FIFO8       |  GN2
                                         GN1
```

The Jedec Data is stored in ===> scope.jed
The Placements are stored in ===> scope.plc
The Design Doc is stored in ===> scope.rpt

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: scope

PLACEMENT FILE

```
Flags Used:              Unplace=False          Max Packing=True
Flags Used:          Expand Small=False          Expand All=False

; FITR generated placements - scope

Pin    33                 AD0 ; Inp         ; I  4
Pin    43                 AD1 ; Inp         ; D  0
Pin    42                 AD2 ; Inp         ; D  2
Pin    41                 AD3 ; Inp         ; D  4
Pin    39                 AD4 ; Inp         ; D  8
Pin    38                 AD5 ; Inp         ; D 10
Pin    35          FAST_CLOCK ; Inp         ; I  5
Pin    20               FIFO0  Comb         ; B  2
Pin    19               FIFO1  Comb         ; B  4
Pin    18               FIFO2  Comb         ; B  6
Pin    17               FIFO3  Comb         ; B  8
Pin     3               FIFO4  Comb         ; A  2
Pin     4               FIFO5  Comb         ; A  4
Pin     5               FIFO6  Comb         ; A  6
Pin     2               FIFO7  Comb         ; A  0
Pin    21               FIFO8  Comb         ; B  0
Pin     7         /FIFO_WRITE   Reg         ; A 10
Pin     6        FRAME_STROBE   Reg         ; A  8
Pin    24                 GN1  Comb         ; C  0
Pin    25                 GN2  Comb         ; C  2
Pin    32              NOTEOC ; Inp         ; I  3
Pin    11   NOT_WRITE_COMMAND ; Inp         ; I  1
Pin    37               RESET ; Inp         ; D 12
Pin    13     SERIAL_CLOCK_IN ; Inp         ; I  2
Pin    36      SERIAL_DATA_IN ; Inp         ; D 14
Pin    10         START_FRAME ; Inp         ; I  0
Node   53                  EH  Comb         ; D  3
Node   64                  EL  Comb         ; D 14
Node   14            EOC_QUAL   Reg         ; A 12
Node   41                 ZZ0   Reg         ; C  7
Node   19                 ZZ1   Reg         ; B  1
Node   21                 ZZ2   Reg         ; B  3
Node   25                 ZZ3   Reg         ; B  7
Node   40                 ZZ4   Reg         ; C  6
Node   50                 ZZ5   Reg         ; D  0
Node   42                 ZZ6   Reg         ; C  8
Node   51                 ZZ7   Reg         ; D  1
Node   49                 ZZ8   Reg         ; C 15
Node   32                 ZZ9   Reg         ; B 14
Node   28                ZZ10   Reg         ; B 10
Node   33                ZZ11   Reg         ; B 15
Node   39                 CH0  Comb         ; C  5
Node   43                 CH1  Comb         ; C  9
Node   37                 CH2  Comb         ; C  3
Node   38                 CH3  Comb         ; C  4
Node    7                 CT0   Reg         ; A  5
Node   11                 CT1   Reg         ; A  9
Node   13                 CT2   Reg         ; A 11
Node   15                 CT3   Reg         ; A 13
Node    3                CTL0   Reg         ; A  1
Node   16                CTL1   Reg         ; A 14
```

```
Node  17            CTL2   Reg        ; A 15
Node   5           START   Reg        ; A  3
;
; Unused Pins & Nodes
;
; -> None Found.
;
;-; Sequence
;-; FIFO7 FIFO4 FIFO5 FIFO6 FRAME_STROBE FIFO_WRITE CTL0 START CT0 CT1 CT2 EOC_QUAL
;-; CT3 CTL1 CTL2 GN1 GN2 CH2 CH3 CH0 ZZ4 ZZ0 ZZ6 CH1 ZZ8 FIFO8 FIFO0 FIFO1 FIFO2
;-; FIFO3 ZZ1 ZZ2 ZZ3 ZZ10 ZZ9 ZZ11 ZZ5 ZZ7 EH EL START_FRAME NOT_WRITE_COMMAND
;-; NOTEOC AD0 AD1 AD2 AD3 AD4 AD5 RESET SERIAL_DATA_IN
;
```

32-Bit CRC Error Detection: MACH210 and MACH230

Advanced Micro Devices

Application Note

by Phil Madison, AT&T-Paradyne and Gene Crossley, AMD, United States

ABSTRACT

The transmission and reception of digital data over local area networks (LANs) is more popular than ever. However, reliable data transmission from one system to another requires an efficient and effective error detection method. The IEEE has chosen the 32-bit cyclic redundancy check (CRC) for its 802.3 Ethernet system.

This application note describes a 32-bit CRC system implemented in two Macro Array CMOS High-speed (MACH) devices from Advanced Micro Devices (AMD). The MACH is an electrically erasable Programmable Logic Device that provides the best of both worlds: the ability to retain a design without power being constantly applied and the ability to be reprogrammed, thus reducing waste and additional cost if a logic change is necessary.

The 32-bit CRC system utilizes one MACH device to house the generation logic and detection logic. This implementation differs from a standard serial CRC system by allowing transmission data to be handled four bits at a time via parallel inputs to the generating and detection circuits. The final CRC that will be appended to the transmission data is also provided in eight bit bytes to allow faster data rates to be supported.

The authors wish to acknowledge the use of several excerpts from a previous AMD application note by Karen Spesard on a standard CRC error detection implementation in a different programmable logic device.

INTRODUCTION

In transferring digital information from one system to another, the possibility of data transmission errors is always present. The Cyclic Redundancy Check or CRC developed for the data communications marketplace is intended to detect the majority of these errors. Utilizing two MACH devices from AMD, a 32-bit CRC implementation is achieved for Ethernet, one of the industry's most popular local area networks. The MACH device speeds will also allow these circuits to be used with higher data transmission rate networks than the 10 Mbits/s Ethernet system, if desired.

CRC OVERVIEW

The Cyclic Redundancy Check was developed by the IEEE for detecting data transmission errors. It is composed of a set of linear feedback shift registers (LSFRs) designed to represent a fixed polynomial G(x). This polynomial is the divisor in the following equation:

$$x^n \frac{D(x)}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

where D(x) is the transmitted data polynomial being checked for errors. The data polynomial is multiplied by the length of the generator polynomial x to always insure the remainder of the equation is different from the data itself.

The data is shifted into the LSFR then divided by G(x). This division produces a quotient Q(x) and remainder R(x). The quotient is ignored, but the Remainder bits R(x) stay in the LSFRs. Thus, the remainder polynomial or redundancy check bits remain in the registers after the completion of shifting of the data. These remainder bits, or check bits, are then inverted and appended to the transmitted bit stream to produce the encoded data stream, D(x) + R(x).

On the receiving end, the data is shifted into a similar LSFR configuration incorporating the same G(x) function. Since modulo-2 addition and subtraction operations are the same, the first equation becomes

$$Q(x)G(x) = x^nD(x) + R(x).$$

Remembering that D(x) was prescaled and appending the remainder to the data is equivalent to adding it, the encoded polynomial should be exactly divisible by G(x). If the operation on the receiving end produces a result of zero (0), it may be assumed that all of the transmitted bits arrived unaltered. Anything other than a zero (0) result implies that a data transmission bit was altered and an error is detected and the error flag should be set.

GENERATING CRC BITS AND CHECKING DATA

An example of the LSFR hardware needed to generate CRC bits in a generic serial fashion for a three-bit G(x) is displayed in Figure 8-1. As shown, the last register output is XORed with the data bit and then fed back to registers X0 and X2. The feedback term positions within the LFSR correspond to all but the highest power of x in the generator polynomial. In this example, the feedback positions are determined from x^2 and x^0.

Publication# 17750 Rev. A Amendment/0
Issue Date: May 1993

The redundancy check-bits can be calculated from Table 1 for $G(x)=x^3+x^2+1$ where the internal state of each register in the LSFR after every shift is shown. Therefore, if the data is 10011, the redundancy check bits are D5=0, D6=1, and D7=0 after the last data bit is shifted in (5th shift). This is because D0=1, D1=1, D2=0, D3=0, and D4=1.

($D(x) = 1 + 0x + 0x^2 + 1x^3 + 1x^4$ and the least significant bit, LSB, is the first bit transmitted.) The check bits are shifted out from the LSFR by disabling the feedback terms of the LSFR. The check bits can be verified by long division.

The encoded data then becomes 010 10011. To check for data integrity, the encoded data is shifted into another LSFR with the same G(x). This time the last bit (8th bit) is shifted in and, if the message was unaltered, the bits residing in the registers are zero. If the redundancy bits are not zero, an error occurred. This can also be verified by long division.

In summary, CRC modifies the data polynomial so that it is exactly divisiable by a fixed polynomial G(x). When the modified polynomial is received, it is checked for the exact division by G(x). If an error occurred, error flag is set.

IEEE 32-BIT CRC DESIGN CRITERIA

CRC error detection is one of the best methods for checking the validity of large frames of information. It can detect all errors within n successive bits, all errors with an odd number of bits in error for an even G(x), and, of course, all error patterns that are not divisible by G(x) .

The IEEE-802.3/Ethernet Local Area Network Standard defines a 32-bit CRC code. This code works with data ranges from 46 to 1500 bytes, and is also used in the Autodin-II Network. It has the capability of checking up to 268 Mbytes of transmitted data if other networks are being considered.

The generator polynomial used for the Ethernet 32 bit CRC is defined as:

$$G(x)=1+x+x^2+x^4+x^5+x^7+x^8+x^{10}+x^{11}+x^{12}+x^{16}+x^{22}+x^{23}+x^{26}+x^{32}$$

where the coefficients of the generator polynomial correspond to the feedback position of the LSFR.

The standard specifies that on the transmitter end, the shift registers are preloaded to ones at the beginning of every new transmission. When all of the data bits have been entered, the complemented CRC is shifted out for transmission and appended to the end of the data stream.

At the receiver, the transmitted data bits and the complemented check bits are received. When the end of the frame flag is reached, the remainder is checked. If no error occurs, the final contents of the CRC shift register are:

X31							X0
1100	0111	0000	0100	1101	1101	0111	1011
C	7	0	4	D	D	7	B

This remainder is not zero because the checkbits are complemented before being transmitted to the receiver.

FOUR-BIT PARALLEL INPUT IMPLEMENTATION

The remainder of this application note deals with a modified implementation of the 32-bit CRC. This design utilizes a parallel input of 4-bits for each shift of the LSFRs. This implementation allows faster data rates to be supported. The input data is presented starting with the first 4 bits being transmitted, the first bit in the D0 position and fourth bit in the D3 position. After the initial shift, the next 4 bits are presented in the same manner and is repeated until the transmission is complete. The transmission data block must be an even multiple of 4 bits.

WHY USE MACH DEVICES IN CRC?

The 32-bit serial CRC transmission and reception sections can be implemented in several ways. One way would be to acquire standard CRC devices. These chips, however, only exist for specific G(x) and therefore can be inflexible, especially for the 4-bit parallel input case that is desired. Another method would be to design with standard PLD devices. This method would require 8 standard 20-pin PLD devices for the dual 32-bit LSFRs, with eight bits in each PLD without the consideration for the control logic required. A third method would be an LCA device but a trade-off of density versus speed may not allow the application to be able to support the data speeds required for the application. The fourth choice, and the one in this application note, is to use a MACH device which give the speed and density needed for this application.

In addition, the MACH devices can provide the designer control of the design after release. Since the parts are electrically eraseable, the design may be changed by reprogramming the part in the programmer. The device may be used again and again which reduces the cost development and the quick erasure in the programmer speeds design change turnarounds.

CRC LOGIC IMPLEMENTATION

Figures 8-2 and 8-3 show the complete design in schematic form for the 32-bit CRC standard generation polynomial with the 4-bit parallel input modification described above. Figure 8-2 depicts the generation circuit and Figure 8-3 the detection circuit. The S generator polynomial is implemented with thirty two

registers in each LSFR. The feedback terms (Q31 xor DIN) are at registers QO, Q1, Q2, ... Q28, and Q29 and correspond to the Ethernet generator polynomial, G(x) given above.

The registers in the LSFRs are D-type flip-flops and are clocked synchronously. In the transmitter portion, 4 pins—INIT, IN0, and IN1—are respectively added for initializing the registers and controlling the proper byte to be placed on the output pins when the generation is completed. When INIT is HIGH, all registers are initialized to HIGH. Table 8-1 depicts the required IN0 and IN1 values for the proper CRC byte to be placed on the pins OUT0–OUT7. OUT0 is the least significant bit (LSB) and OUT7 is the most significant bit (MSB) of the byte selected.

Table 8-1. Byte Selection for the Generator Output

IN1	IN0	Byte Selected
0	0	BYTE 0 (Bits 0–7)
0	1	BYTE 1 (Bits 8–15)
1	0	BYTE 2 (Bits 16–23)
1	1	BYTE 3 (Bits 24–31)

The receiver LSFR is essentially the same as the transmitter LSFR. However, after data and the thirty two check bits are received, the data is checked for errors. If an error occurred, the error flag is set. Otherwise, the data reception is considered to be error-free.

The generation and detection circuits are both implemented with a MACH230 device. The MACH230 provides 128 macrocells, each containing a flip-flop that may be configured as a D, T flip-flop, or a transparent latch. The macrocell may also be configured as combinatorial only and the polarity is selectable. Each macrocell has a group of 4 AND_OR product terms associated with it and can borrow up to 12 additional product terms from adjacent macrocells to give a maximum of 16 product terms per macrocell. In addition to the macro cells, 64 I/O cells attached to device pins and 6 additional dedicated input pins provide access to and from the device. The macrocells may be "buried" internally, i.e., have no contact with an external pin through an I/O cell but only provide feedback to the internal array. Additionally, the macrocell may be attached to an I/O cell and provide an output to an external pin. The macrocells are connected internally via a fuse map array and a switch matrix to provide routing. The switch matrix is designed and implemented in a manner that provides minimum delay through the device and allows the propogation delay to be guaranteed up to the maximum number of product terms per output.

Due to the speed required for the design and the desire to only have a single pass through the device for each shift, four pre-terms were developed to make the total number of necessary registers for each section to 36. These pre-terms were combinations of the four input bits and four of the LSFR registers that were common terms throughout the LSFR chain. They are clocked with a different clock signal to allow these locations to be changed on one edge of the data clock and the LSFRs to be updated on the opposite edge of the clock. This pipeline arrangement permits a faster transmission data rate.

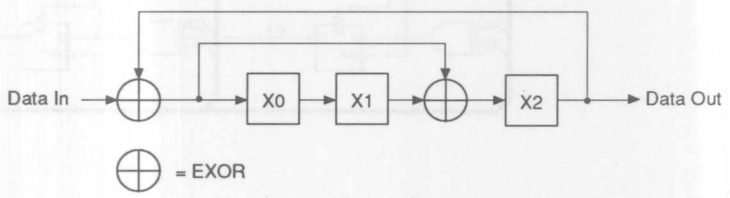
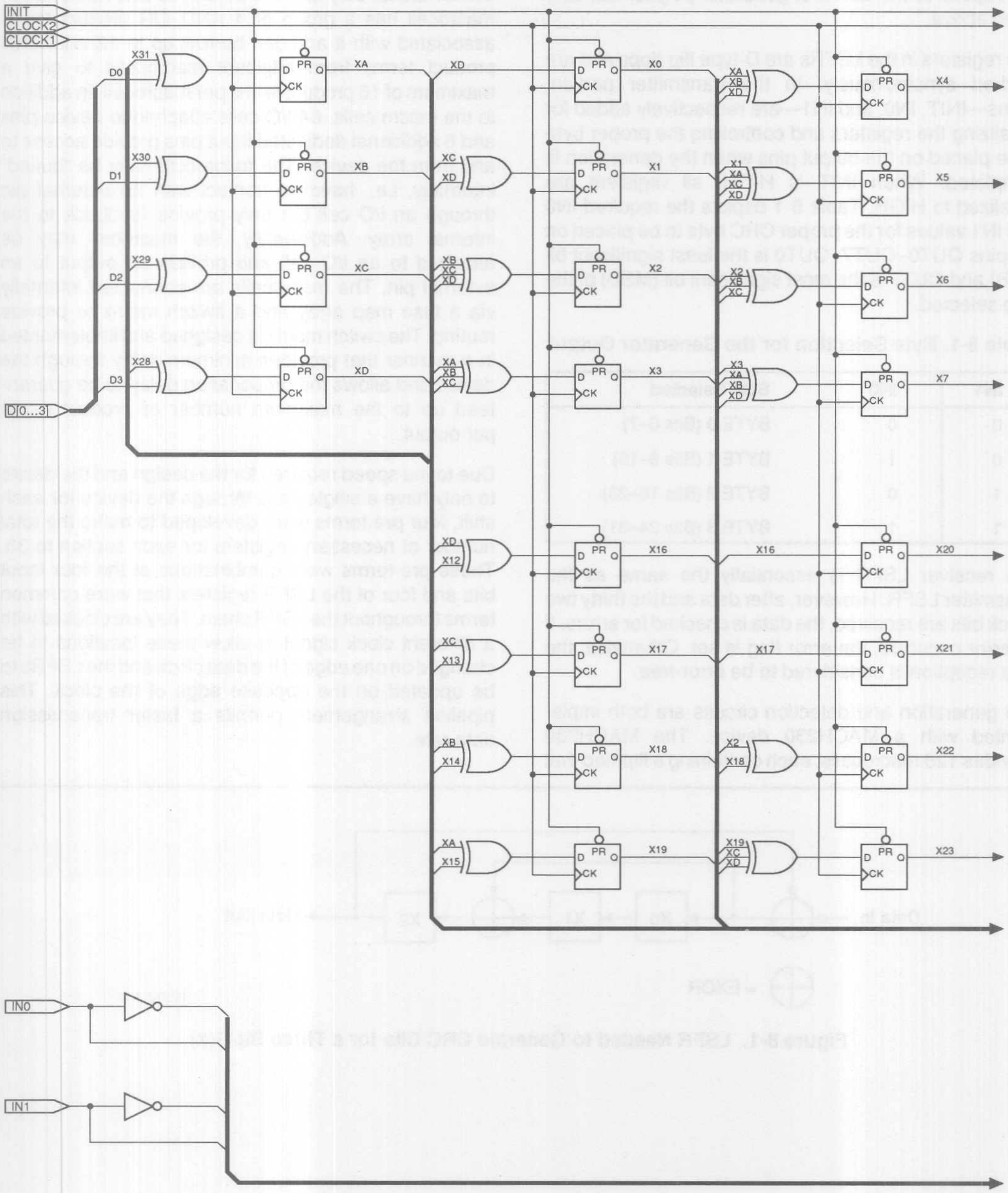


Figure 8-1. LSFR Needed to Generate CRC Bits for a Three Bit G(x)



17020A-030A

Figure 8-2. CRC Generator

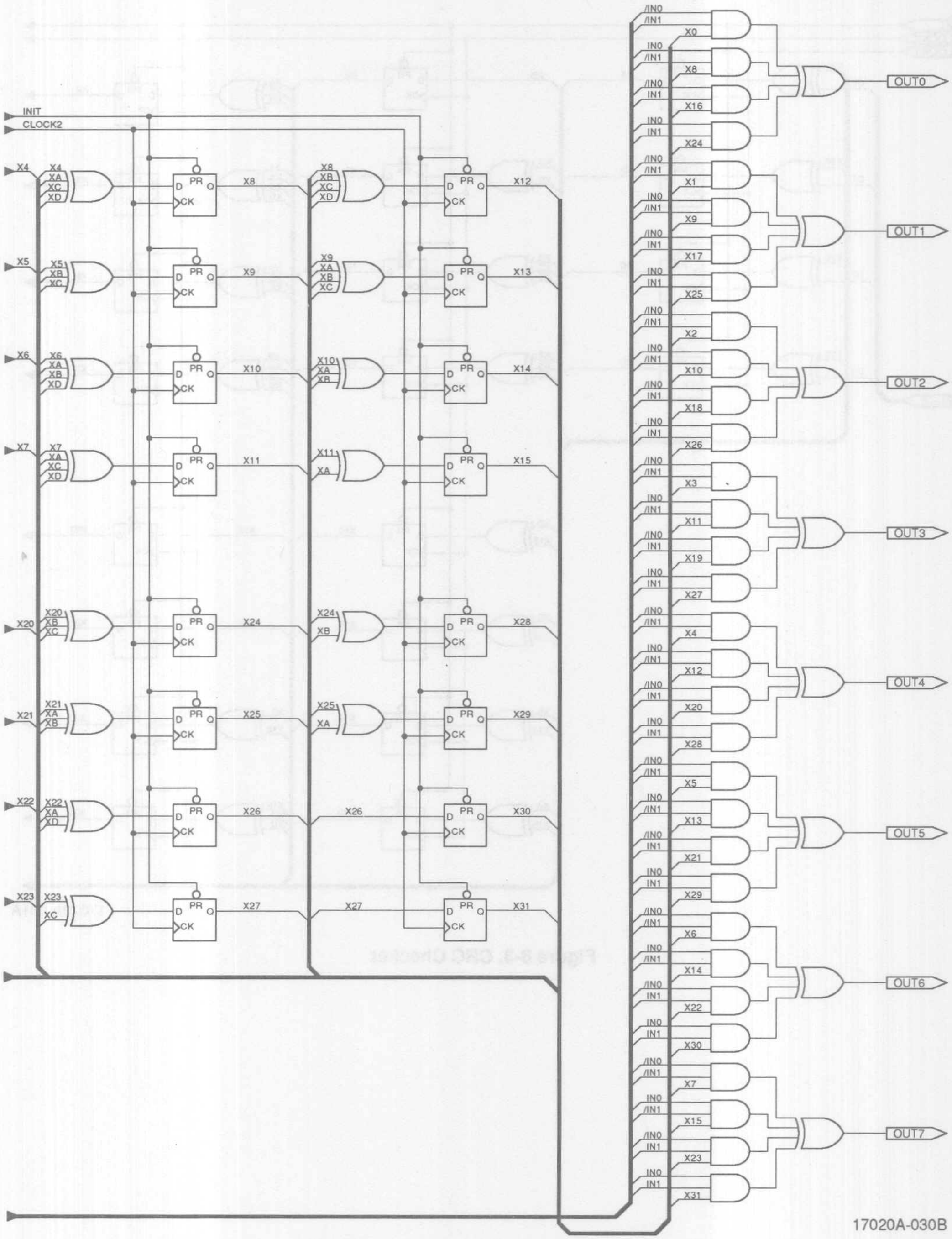


Figure 8-2. CRC Generator (continued)

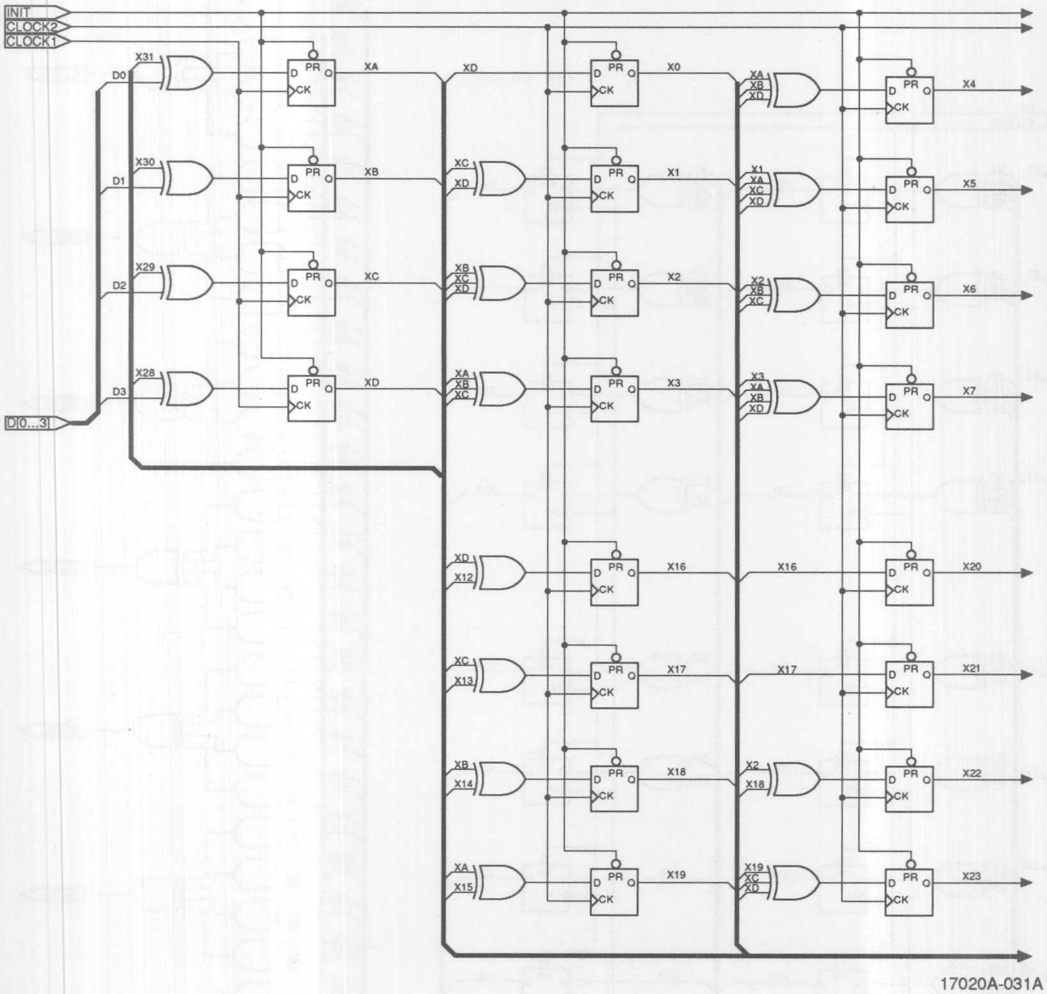


Figure 8-3. CRC Checker

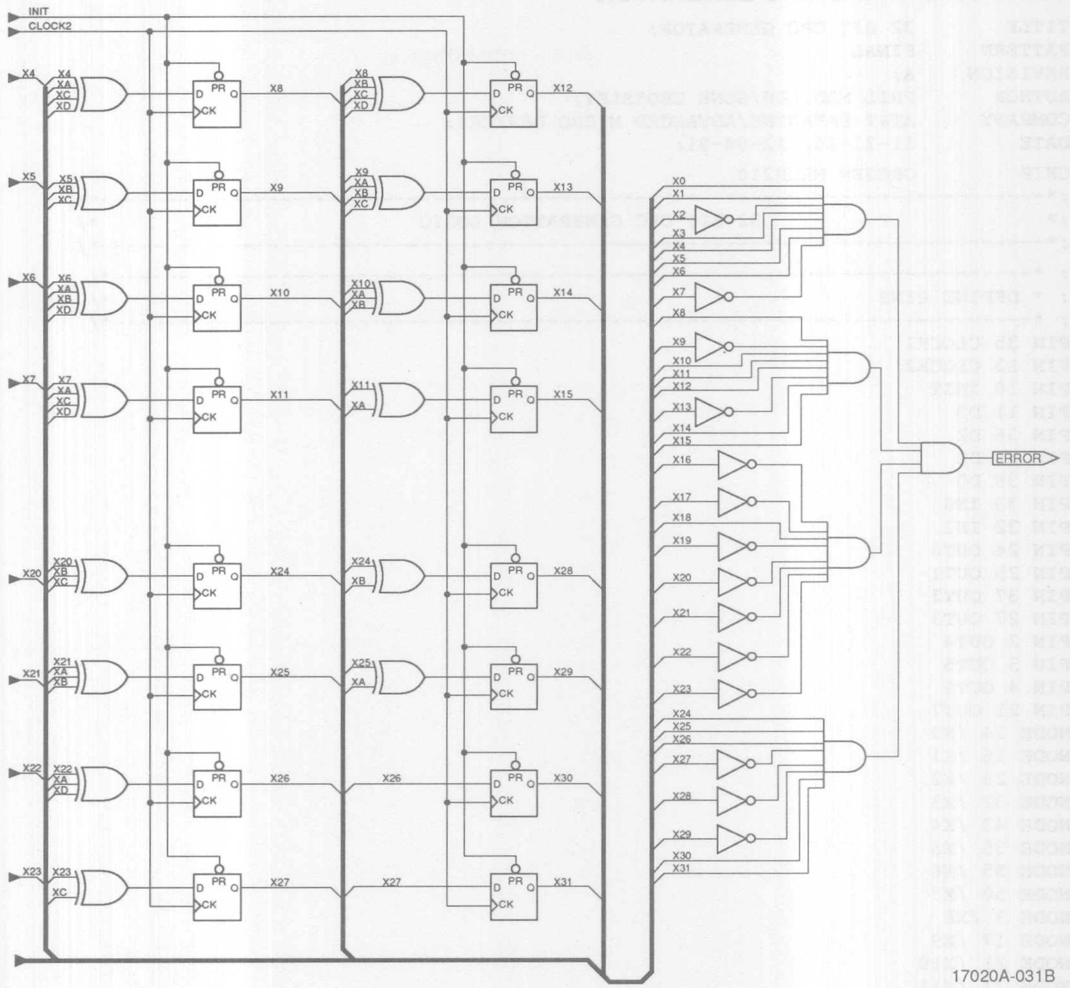


Figure 8-3. CRC Checker (continued)

DESIGN FILE: MACH210 GENERATOR

```
TITLE       32 BIT CRC GENERATOR;
PATTERN     FINAL
REVISION    A;
AUTHOR      PHIL MADISON/GENE CROSSLEY;
COMPANY     AT&T PARADYNE/ADVANCED MICRO DEVICES;
DATE        11-21-90, 12-04-91;
CHIP        CRCGEN MACH210
;*------------------------------------------------------------------------------*/
;*                     32 BIT CRC GENERATION LOGIC                              */
;*------------------------------------------------------------------------------*/

; *-----------------------------------------------------------------------------*/
; * DEFINE PINS                                                                 */
; *-----------------------------------------------------------------------------*/
PIN 35 CLOCK1
PIN 13 CLOCK2
PIN 10 INIT
PIN 11 D3
PIN 36 D2
PIN 38 D1
PIN 39 D0
PIN 33 IN0
PIN 32 IN1
PIN 24 OUT0
PIN 25 OUT1
PIN 37 OUT2
PIN 27 OUT3
PIN 2 OUT4
PIN 3 OUT5
PIN 4 OUT6
PIN 21 OUT7
NODE 14 /X0
NODE 16 /X1
NODE 29 /X2
NODE 32 /X3
NODE 43 /X4
NODE 35 /X5
NODE 55 /X6
NODE 50 /X7
NODE 3 /X8
NODE 17 /X9
NODE 21 /X10
NODE 31 /X11
NODE 39 /X12
NODE 41 /X13
NODE 53 /X14
NODE 54 /X15
NODE 10 /X16
NODE 12 /X17
NODE 24 /X18
NODE 28 /X19
NODE 47 /X20
NODE 48 /X21
NODE 65 /X22
NODE 51 /X23
NODE 7 /X24
NODE 9 /X25
NODE 20 /X26
NODE 22 /X27
NODE 42 /X28
NODE 46 /X29
NODE 60 /X30
```

```
NODE 61 /X31
NODE 56 XA
NODE 58 XB
NODE 59 XC
NODE 23 XD
GROUP MACH_SEG_A X0 X8 X16 X24 X1 X9 X17 X25
GROUP MACH_SEG_B X2 X10 X18 X26 X3 X11 X19 X27
GROUP MACH_SEG_C X4 X12 X20 X28 X5 X13 X21 X29
GROUP MACH_SEG_D X6 X14 X22 X30 X7 X15 X23 X31
;*---------------------------------------------------------------------------*/
;* DEFINE BOOLEAN EQUATIONS                                                  */
;*---------------------------------------------------------------------------*/
EQUATIONS
XA      := D0:+:X31
XA.CLKF = CLOCK2
XA.RSTF = INIT
XA.SETF = GND
XB      := D1:+:X30
XB.CLKF = CLOCK2
XB.RSTF = INIT
XB.SETF = GND
XC      := D2:+:X29
XC.CLKF = CLOCK2
XC.RSTF = INIT
XC.SETF = GND
XD      := D3:+:X28
XD.CLKF = CLOCK2
XD.RSTF = INIT
XD.SETF = GND
X0      := XD
X0.CLKF = CLOCK1
X0.RSTF = INIT
X0.SETF = GND

X1      := XD:+:XC
X1.CLKF = CLOCK1
X1.RSTF = INIT
X1.SETF = GND

X2      := XD:+:XC:+:XB
X2.CLKF = CLOCK1
X2.RSTF = INIT
X2.SETF = GND

X3      := XC:+:XB:+:XA
X3.CLKF = CLOCK1
X3.RSTF = INIT
X3.SETF = GND

X4      := XD:+:XB:+:XA
X4.CLKF = CLOCK1
X4.RSTF = INIT
X4.SETF = GND

X5      := XD:+:XA:+:XC:+:X1
X5.CLKF = CLOCK1
X5.RSTF = INIT
X5.SETF = GND

X6      := XC:+:XB:+:X2
X6.CLKF = CLOCK1
X6.RSTF = INIT
```

```
X6.SETF = GND

X7      := XD:+:XB:+:XA:+:X3
X7.CLKF = CLOCK1
X7.RSTF = INIT
X7.SETF = GND

X8      := XD:+:XC:+:XA:+:X4
X8.CLKF = CLOCK1
X8.RSTF = INIT
X8.SETF = GND

X9      := XC:+:XB:+:X5
X9.CLKF = CLOCK1
X9.RSTF = INIT
X9.SETF = GND

X10      := XD:+:XB:+:XA:+:X6
X10.CLKF = CLOCK1
X10.RSTF = INIT
X10.SETF = GND

X11      := XC:+:XA:+:X7:+:XD
X11.CLKF = CLOCK1
X11.RSTF = INIT
X11.SETF = GND

X12      := XD:+:XB:+:X8:+:XC
X12.CLKF = CLOCK1
X12.RSTF = INIT
X12.SETF = GND

X13      := XC:+:XA:+:X9:+:XB
X13.CLKF = CLOCK1
X13.RSTF = INIT
X13.SETF = GND

X14      := XB:+:X10:+:XA
X14.CLKF = CLOCK1
X14.RSTF = INIT
X14.SETF = GND

X15      := XA:+:X11
X15.CLKF = CLOCK1
X15.RSTF = INIT
X15.SETF = GND

X16      := XD:+:X12
X16.CLKF = CLOCK1
X16.RSTF = INIT
X16.SETF = GND

X17      := XC:+:X13
X17.CLKF = CLOCK1
X17.RSTF = INIT
X17.SETF = GND

X18      := XB:+:X14
X18.CLKF = CLOCK1
X18.RSTF = INIT
X18.SETF = GND

X19      := XA:+:X15
X19.CLKF = CLOCK1
```

```
X19.RSTF = INIT
X19.SETF = GND

X20      := X16;
X20.CLKF = CLOCK1
X20.RSTF = INIT
X20.SETF = GND

X21      := X17;
X21.CLKF = CLOCK1
X21.RSTF = INIT
X21.SETF = GND

X22      := XD:+:X18
X22.CLKF = CLOCK1
X22.RSTF = INIT
X22.SETF = GND

X23      := XD:+:XC:+:X19
X23.CLKF = CLOCK1
X23.RSTF = INIT
X23.SETF = GND

X24      := XC:+:XB:+:X20
X24.CLKF = CLOCK1
X24.RSTF = INIT
X24.SETF = GND

X25      := XB:+:XA:+:X21
X25.CLKF = CLOCK1
X25.RSTF = INIT
X25.SETF = GND

X26      := XD:+:XA:+:X22
X26.CLKF = CLOCK1
X26.RSTF = INIT
X26.SETF = GND

X27      := XC:+:X23
X27.CLKF = CLOCK1
X27.RSTF = INIT
X27.SETF = GND

X28      := XB:+:X24
X28.CLKF = CLOCK1
X28.RSTF = INIT
X28.SETF = GND

X29      := XA:+:X25
X29.CLKF = CLOCK1
X29.RSTF = INIT
X29.SETF = GND

X30      := X26
X30.CLKF = CLOCK1
X30.RSTF = INIT
X30.SETF = GND

X31      := X27
X31.CLKF = CLOCK1
X31.RSTF = INIT
X31.SETF = GND
```

```
OUT0 = /IN0*/IN1*X0 +
       /IN0*IN1*X8  +
        IN0*/IN1*X16 +
        IN0*IN1*X24
OUT1 = /IN0*/IN1*X1 +
       /IN0*IN1*X9  +
        IN0*/IN1*X17 +
        IN0*IN1*X25
OUT2 = /IN0*/IN1*X2 +
       /IN0*IN1*X10 +
        IN0*/IN1*X18 +
        IN0*IN1*X26
OUT3 = /IN0*/IN1*X3 +
       /IN0*IN1*X11 +
        IN0*/IN1*X19 +
        IN0*IN1*X27
OUT4 = /IN0*/IN1*X4 +
       /IN0*IN1*X12 +
        IN0*/IN1*X20 +
        IN0*IN1*X28
OUT5 = /IN0*/IN1*X5 +
       /IN0*IN1*X13 +
        IN0*/IN1*X21 +
        IN0*IN1*X29
OUT6 = /IN0*/IN1*X6 +
       /IN0*IN1*X14 +
        IN0*/IN1*X22 +
        IN0*IN1*X30
OUT7 = /IN0*/IN1*X7 +
       /IN0*IN1*X15 +
        IN0*/IN1*X23 +
        IN0*IN1*X31
;*------------------------------------------------------------------------------*/
; SIMULATION SEGMENT
;*------------------------------------------------------------------------------*/
SIMULATION
;*------------------------------------------------------------------------------*/
```

FITTER FILE: MACH210 GENERATOR

Note: This file has been condensed in order to save trees.

```
PALASM 4 v1.2 MACH FITR - MARKET RELEASE (6-7-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990,1991
Flags Used:         Unplace=False          Max Packing=True
Flags Used:       Expand Small=True         Expand All=True
Reading Device Database ...
***************************************************************
     Mach PLD Fitter - v 1.51           32 BIT CRC GENERATOR
***************************************************************
*** Timing Analysis for Signals
 Parameter   Min   Max        Signal List (Those having Max delay.)
      Tpd      1     1        OUT4               OUT5               OUT6
                              OUT7               OUT0               OUT1
                              OUT3               OUT2
      Tsu      1     1        X8                 X24                X25
                              X16                X17                X0
                              X1                 XA
      Tco      0     1        OUT0               OUT1               OUT2
                              OUT3               OUT5               OUT4
                              OUT7               OUT6
      Tcr      1     1        X12                X28                X29
                              X20                X21                X5
                              X13                X26

.
 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - clock to output (register output to combinatorial output)
 Tcr - Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
 All delay values are expressed in terms of array passes
*** Device Resource Checks
               Available         Used          Remaining
        Clocks:     2             2               0
          Pins:    38            17              21       ->    44%
     I/O Macro:    32             8              24
   Total Macro:    64            44              20
 Product Terms:   256           161              52       ->    78%

MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
*** Last Equations Placed in Blocks
Weakly -
*** Block Partitioning Results
               Array     Macros       # I/O     Buried   Product     Signal
               Inputs    Remain       Macro     Logic    Terms       Fanout
   Block-> A     19        5            3          8       48          8
   Block-> B     16        6            1          9       48         16
   Block-> C     19        5            3          8       56         10
   Block-> D     21        4            1         11       52         25
*** Block Signal List
Block-> A               X9               X1                X0             X17
                       X16              X25               X24              X8
                       OUT6             OUT5              OUT4
Block-> B               X3              X11                X2             X19
                       X18               XD               X27             X10
                       X26             OUT7
```

```
Block-> C            X21             X20             X29              X4
                     X28             X13             X12              X5
                    OUT3            OUT1            OUT0

Block-> D            X22             X31             X30              XC
                      XB              XA              X6             X15
                     X14             X23              X7            OUT2

|> INFORMATION F050 - Device Utilization....... *: 69 %

*** Feedback Map - 32 BIT CRC GENERATOR
 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
         | 0|       XB : 0|     |21| /X20           /X23 : 0|     |21|
         | 1|     /X29 : 1|     |20| XC               XC : 1|     |20|
         | 2|     /X14 : 2|     |19| /X12           /X14 : 2|     |19|
         | 3|     /X13 : 3|     |18|                /X15 : 3|     |18| /X31
         | 4|      IN0 : 4|     |17| /X4             /X6 : 4|     |17|
         | 5|       XA : 5|     |16|                  XA : 5|     |16| D3
         ’--’     /X28 : 6|     |15: IN1            /X28 : 6|     |15: IN1
                  /X22 : 7|     |14: /X6            /X22 : 7|     |14|
                   /X5 : 8|     |13|                 /X7 : 8|     |13: XD
                    XD : 9|     |12: INIT             XB : 9|     |12|
                  /X30 :10|     |11: /X21           INIT :10|     |11: IN0
                       ’--+--C--+--’                     ’--+--D--+--’
                    XB : 0|     |21| /X1              XB : 0|     |21| /X26
                    XC : 1|     |20| /X25             XC : 1|     |20| /X29
                   /X8 : 2|     |19| /X0              D0 : 2|     |19|
                   IN1 : 3|     |18|                  D1 : 3|     |18| /X31
                  /X19 : 4|     |17|                /X19 : 4|     |17| D2
                    XA : 5|     |16| /X3              XA : 5|     |16| /X3
                  /X24 : 6|     |15|                 IN1 : 6|     |15: /X11
                   IN0 : 7|     |14: /X17           /X10 : 7|     |14: /X18
                  /X27 : 8|     |13: /X16           /X27 : 8|     |13: /X2
                    XD : 9|     |12: INIT             XD : 9|     |12: INIT
                  /X11 :10|     |11: /X9            /X30 :10|     |11: IN0
                       ’--+-u--u+--’                     ’--+-u--u+--’

*** Logic Map - 32 BIT CRC GENERATOR
 Gbl Inp .--.     I/O  .--+--A--+--.  I/O           I/O  .--+--B--+--.  I/O
     INIT| 0|     OUT4 | 0| 4   |21|                OUT7 | 0| 4   |21|
       D3| 1|      /X8 | 1| 8   |20|                     | 1| *   |20|
   CLOCK2| 2|     OUT5 | 2| 4   |19|                /X26 | 2| 4   |19|
      IN1| 3|          | 3| *   |18|                /X10 | 3| 8   |18|
      IN0| 4|     OUT6 | 4| 4   |17|                /X27 | 4| 2   |17|
   CLOCK1| 5|     /X24 | 5| 4   |16|                  XD | 5| 2   |16|
         ’--’          | 6| .  4|15| /X9            /X18 | 6| 2  .|15|
                  /X25 | 7| 4  2|14| /X1                 | 7| .  4|14| /X3
                  /X16 | 8| 2  .|13|                     | 8| .  8|13| /X11
                       | 9| .  1|12| /X0                 | 9| .  *|12|
                  /X17 |10| 2  .|11|                /X19 |10| 2  4|11| /X2
                       ’--+--C--+--’                     ’--+--D--+--’
                  OUT0 | 0| 4   |21|                 /X7 | 0| 8   |21|
                   /X5 | 1| 8   |20|                /X23 | 1| 4   |20|
                  OUT1 | 2| 4   |19|                     | 2| *   |19|
                       | 3| *   |18|                /X14 | 3| 4   |18|
                       | 4| *   |17|                /X15 | 4| 2   |17|
                  /X12 | 5| 8   |16|                 /X6 | 5| 4   |16|
                  OUT3 | 6| 4  .|15|                  XA | 6| 2  2|15| /X22
                  /X13 | 7| 8  1|14| /X21                | 7| .  .|14|
                  /X28 | 8| 2  1|13| /X20             XB | 8| 2  .|13|
                   /X4 | 9| 4  2|12| /X29             XC | 9| 2  4|12| OUT2
                       |10| *  .|11|                /X30 |10| 1  1|11| /X31
                       ’--+-u--u+--’                     ’--+-u--u+--’
```

```
*** Pin Map - 32 BIT CRC GENERATOR

                                OUT4
                                OUT5  |        .
                             OUT6  |  |        |  .
                             .  |  |  |        |  |  .
                          .  |  |  |  |        |  |  |  .
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
                    | 7                                   39|D0
                    | 8                  G  V             38|D1
                    | 9                  n  c             37|OUT2
                INIT|10                  d  c              36|D2
                  D3|11                                    35|CLOCK1
                Gnd |12            MACH-210                34| Gnd
              CLOCK2|13                                    33|IN0
                    |14             V  G                   32|IN1
                    |15             c  n                   31|
                    |16             c  d                   30|
                    |17                                    29|
                    |   1  1  2  2  2  2  2  2  2  2  2      |
                    |   8  9  0  1  2  3  4  5  6  7  8      |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                        '  |  |  |        |  |  |  |  '
                           '  |  |        |  |  |  OUT3
                              '  |        |  |  '
                              OUT7        |  OUT1
                                          OUT0
The Design Doc is  stored in ===> 4t32crca.Rpt
The Jedec Data is  stored in ===> 4t32crca.Jed
The Placements are stored in ===> 4t32crca.Plc

%% FITR %% Error Count: 0, Warning Count: 0
%% FITR %% File Processed Successfully. - File: 4t32crca.pds
```

PLACEMENT FILE: MACH210 GENERATOR

```
; Flags Used:            Unplace=False          Max Packing=True
; Flags Used:         Expand Small=True          Expand All=True
; FITR generated placements
Pin     2         OUT4   Comb          ; A  0
Pin     3         OUT5   Comb          ; A  2
Pin     4         OUT6   Comb          ; A  4
Pin    10         INIT ; Inp           ; I  0
Pin    11           D3 ; Inp           ; I  1
Pin    13       CLOCK2 ; Inp           ; I  2
Pin    21         OUT7   Comb          ; B  0
Pin    24         OUT0   Comb          ; C  0
Pin    25         OUT1   Comb          ; C  2
Pin    27         OUT3   Comb          ; C  6
Pin    32          IN1 ; Inp           ; I  3
Pin    33          IN0 ; Inp           ; I  4
Pin    35       CLOCK1 ; Inp           ; I  5
Pin    36           D2 ; Inp           ; D 14
Pin    37         OUT2   Comb          ; D 12
Pin    38           D1 ; Inp           ; D 10
Pin    39           D0 ; Inp           ; D  8
Node    3          /X8    Reg          ; A  1
Node    7         /X24    Reg          ; A  5
Node    9         /X25    Reg          ; A  7
Node   10         /X16    Reg          ; A  8
Node   12         /X17    Reg          ; A 10
Node   14          /X0    Reg          ; A 12
Node   16          /X1    Reg          ; A 14
Node   17          /X9    Reg          ; A 15
Node   20         /X26    Reg          ; B  2
Node   21         /X10    Reg          ; B  3
Node   22         /X27    Reg          ; B  4
Node   23           XD    Reg          ; B  5
Node   24         /X18    Reg          ; B  6
Node   28         /X19    Reg          ; B 10
Node   29          /X2    Reg          ; B 11
Node   31         /X11    Reg          ; B 13
Node   32          /X3    Reg          ; B 14
Node   35          /X5    Reg          ; C  1
Node   39         /X12    Reg          ; C  5
Node   41         /X13    Reg          ; C  7
Node   42         /X28    Reg          ; C  8
Node   43          /X4    Reg          ; C  9
Node   46         /X29    Reg          ; C 12
Node   47         /X20    Reg          ; C 13
Node   48         /X21    Reg          ; C 14
Node   50          /X7    Reg          ; D  0
Node   51         /X23    Reg          ; D  1
Node   53         /X14    Reg          ; D  3
Node   54         /X15    Reg          ; D  4
Node   55          /X6    Reg          ; D  5
Node   56           XA    Reg          ; D  6
Node   58           XB    Reg          ; D  8
Node   59           XC    Reg          ; D  9
Node   60         /X30    Reg          ; D 10
Node   61         /X31    Reg          ; D 11
Node   65         /X22    Reg          ; D 15
```

DESIGN FILE: MACH210 CHECKER

```
TITLE        32 BIT CRC CHECKER
PATTERN      FINAL
REVISION     A;
AUTHOR       PHIL MADISON/GENE CROSSLEY
COMPANY      AT&T PARADYNE/ADVANCED MICRO DEVICES
DATE         11-21-90, 12-04-91;
CHIP         CRCCHK MACH210
;*-------------------------------------------------------------------------*/
;*                           32 BIT CRC CHECK LOGIC                        */
;*-------------------------------------------------------------------------*/
; *-----------------------------------------------------------------------*/
; * DEFINE PINS                                                           */
; *-----------------------------------------------------------------------*/
PIN 35 CLOCK1
PIN 13 CLOCK2
PIN 10 INIT
PIN 11 D3
PIN 36 D2
PIN 38 D1
PIN 39 D0
PIN 26 ERRFLG
NODE 14 X0
NODE 16 X1
NODE 29 X2
NODE 32 X3
NODE 43 X4
NODE 35 X5
NODE 55 X6
NODE 50 X7
NODE 3 X8
NODE 17 X9
NODE 21 X10
NODE 31 X11
NODE 39 X12
NODE 41 X13
NODE 53 X14
NODE 54 X15
NODE 10 X16
NODE 12 X17
NODE 24 X18
NODE 28 X19
NODE 47 X20
NODE 48 X21
NODE 65 X22
NODE 51 X23
NODE 7 X24
NODE 9 X25
NODE 20 X26
NODE 22 X27
NODE 42 X28
NODE 46 X29
NODE 60 X30
NODE 61 X31
NODE 5 E1
NODE 26 E2
NODE 37 E3
NODE 57 E4
NODE 56 XA
NODE 58 XB
NODE 59 XC
NODE 23 XD
;*-------------------------------------------------------------------------*/
;* DEFINE BOOLEAN EQUATIONS                                                */
;*-------------------------------------------------------------------------*/
EQUATIONS
XA      := D0:+:X31
XA.CLKF = CLOCK2
```

```
XA.RSTF = GND
XA.SETF = INIT
XB      := D1:+:X30
XB.CLKF = CLOCK2
XB.RSTF = GND
XB.SETF = INIT
XC      := D2:+:X29
XC.CLKF = CLOCK2
XC.RSTF = GND
XC.SETF = INIT
XD      := D3:+:X28
XD.CLKF = CLOCK2
XD.RSTF = GND
XD.SETF = INIT
X0      := XD
X0.CLKF = CLOCK1
X0.RSTF = GND
X0.SETF = INIT

X1      := XD:+:XC
X1.CLKF = CLOCK1
X1.RSTF = GND
X1.SETF = INIT

X2      := XD:+:XC:+:XB
X2.CLKF = CLOCK1
X2.RSTF = GND
X2.SETF = INIT

X3      := XC:+:XB:+:XA
X3.CLKF = CLOCK1
X3.RSTF = GND
X3.SETF = INIT

X4      := XD:+:XB:+:XA
X4.CLKF = CLOCK1
X4.RSTF = GND
X4.SETF = INIT

X5      := XD:+:XA:+:XC:+:X1
X5.CLKF = CLOCK1
X5.RSTF = GND
X5.SETF = INIT

X6      := XC:+:XB:+:X2
X6.CLKF = CLOCK1
X6.RSTF = GND
X6.SETF = INIT

X7      := XD:+:XB:+:XA:+:X3
X7.CLKF = CLOCK1
X7.RSTF = GND
X7.SETF = INIT

X8      := XD:+:XC:+:XA:+:X4
X8.CLKF = CLOCK1
X8.RSTF = GND
X8.SETF = INIT

X9      := XC:+:XB:+:X5
X9.CLKF = CLOCK1
X9.RSTF = GND
```

```
X9.SETF = INIT

X10      := XD:+:XB:+:XA:+:X6
X10.CLKF = CLOCK1
X10.RSTF = GND
X10.SETF = INIT

X11      := XC:+:XA:+:X7:+:XD
X11.CLKF = CLOCK1
X11.RSTF = GND
X11.SETF = INIT

X12      := XD:+:XB:+:X8:+:XC
X12.CLKF = CLOCK1
X12.RSTF = GND
X12.SETF = INIT

X13      := XC:+:XA:+:X9:+:XB
X13.CLKF = CLOCK1
X13.RSTF = GND
X13.SETF = INIT

X14      := XB:+:X10:+:XA
X14.CLKF = CLOCK1
X14.RSTF = GND
X14.SETF = INIT

X15      := XA:+:X11
X15.CLKF = CLOCK1
X15.RSTF = GND
X15.SETF = INIT

X16      := XD:+:X12
X16.CLKF = CLOCK1
X16.RSTF = GND
X16.SETF = INIT

X17      := XC:+:X13
X17.CLKF = CLOCK1
X17.RSTF = GND
X17.SETF = INIT

X18      := XB:+:X14
X18.CLKF = CLOCK1
X18.RSTF = GND
X18.SETF = INIT

X19      := XA:+:X15
X19.CLKF = CLOCK1
X19.RSTF = GND
X19.SETF = INIT

X20      := X16;
X20.CLKF = CLOCK1
X20.RSTF = GND
X20.SETF = INIT

X21      := X17;
X21.CLKF = CLOCK1
X21.RSTF = GND
X21.SETF = INIT

X22      := XD:+:X18
X22.CLKF = CLOCK1
```

```
X22.RSTF = GND
X22.SETF = INIT

X23      := XD:+:XC:+:X19
X23.CLKF = CLOCK1
X23.RSTF = GND
X23.SETF = INIT

X24      := XC:+:XB:+:X20
X24.CLKF = CLOCK1
X24.RSTF = GND
X24.SETF = INIT

X25      := XB:+:XA:+:X21
X25.CLKF = CLOCK1
X25.RSTF = GND
X25.SETF = INIT

X26      := XD:+:XA:+:X22
X26.CLKF = CLOCK1
X26.RSTF = GND
X26.SETF = INIT

X27      := XC:+:X23
X27.CLKF = CLOCK1
X27.RSTF = GND
X27.SETF = INIT

X28      := XB:+:X24
X28.CLKF = CLOCK1
X28.RSTF = GND
X28.SETF = INIT

X29      := XA:+:X25
X29.CLKF = CLOCK1
X29.RSTF = GND
X29.SETF = INIT

X30      := X26
X30.CLKF = CLOCK1
X30.RSTF = GND
X30.SETF = INIT

X31      := X27
X31.CLKF = CLOCK1
X31.RSTF = GND
X31.SETF = INIT
E1 = X5+/X12+/X13+X4+/X20+/X21
E2 = X28+/X7+/X23+X14+X15+X6+/X22
E3 = X0+X8+X24+X25+/X16+/X17+X1+/X9
E4 = X26+X10+/X27+X18+/X19+/X2+X11+X3+/X29+X30+X31
ERRFLG      := E1+E2+E3+E4
ERRFLG.CLKF = CLOCK2
ERRFLG.RSTF = GND
ERRFLG.SETF = INIT
;*----------------------------------------------------------------------------*/
; SIMULATION SEGMENT
;*----------------------------------------------------------------------------*/
SIMULATION
;*----------------------------------------------------------------------------*/
```

FITTER FILE: MACH210 CHECKER

Note: This file has been condensed in order to save trees.

```
PALASM 4 v1.2 MACH FITR - MARKET RELEASE (6-7-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990,1991
Flags Used:             Unplace=False           Max Packing=False
Flags Used:          Expand Small=True          Expand All=True
Reading Device Database ...
**************************************************************
     Mach PLD Fitter - v 1.51           32 BIT CRC CHECKER
**************************************************************
*** Timing Analysis for Signals
 Parameter   Min   Max          Signal List (Those having Max delay.)
      Tsu     1     1          ERRFLG          X8              X24
                                  X25          X16             X17
                                   X0           XA
      Tco     0     1              E3           E4              E1
                                   E2
      Tcr     1     2          ERRFLG
.
 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - clock to output (register output to combinatorial output)
 Tcr - Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                Available        Used         Remaining
        Clocks:     2              2              0
          Pins:    38              8             30    ->    21%
     I/O Macro:    32              1             31
   Total Macro:    64             41             23
 Product Terms:   256            134             64    ->    74%

MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
*** Last Equations Placed in Blocks
Weakly -
*** Block Partitioning Results
               Array    Macros      # I/O    Buried   Product    Signal
               Inputs   Remain      Macro    Logic     Terms     Fanout
   Block-> A     11        7          0         9        40         9
   Block-> B     13        6          0        10        48        13
   Block-> C     17        6          1         9        52         9
   Block-> D     19        4          0        12        52        21
*** Block Signal List
Block-> A               X9             X1             X0             X17
                       X16            X25            X24              E1
                        X8
Block-> B               X3            X11             X2             X19
                        E2            X18             XD             X27
                       X10            X26
Block-> C              X21            X20            X29              X4
                       X28            X13            X12              E3
                        X5         ERRFLG
```

```
Block-> D             X22             X31             X30               XC
                       XB              E4              XA               X6
                      X15             X14             X23               X7

|> INFORMATION F050 - Device Utilization....... *: 53 %
 *** Feedback Map - 32 BIT CRC CHECKER
 Gbl Inp .--.      I/O  .--+--A--+--.  I/O            I/O  .--+--B--+--.  I/O
         | 0|        XB : 0|     |21|                  X23 : 0|     |21|
         | 1|        XC : 1|     |20|                   XC : 1|     |20|
         | 2|       X20 : 2|     |19| X12              X14 : 2|     |19|
         | 3|       X13 : 3|     |18|                  X15 : 3|     |18|
         | 4|           | 4|     |17|                   X6 : 4|     |17|
         | 5|        XA : 5|     |16| XD                XA : 5|     |16| D3
         '--'           | 6|     |15|                  X28 : 6|     |15|
                        | 7|     |14|                  X22 : 7|     |14|
                     X5 : 8|     |13|                   X7 : 8|     |13: XD
                     X4 : 9|     |12|                   XB : 9|     |12|
                   INIT :10|     |11: X21             INIT :10|     |11|
                        '--+--C--+--'                      '--+--D--+--'
                     XB : 0|     |21| X1                XB : 0|     |21| X26
                     XC : 1|     |20|                   XC : 1|     |20| X29
                     X8 : 2|     |19| X0               X30 : 2|     |19|
                     E2 : 3|     |18| X24               D1 : 3|     |18| X31
                     E1 : 4|     |17|                  X19 : 4|     |17| D2
                     XA : 5|     |16|                   XA : 5|     |16| X3
                     E4 : 6|     |15|                      | 6|     |15|
                        | 7|     |14: X17              X10 : 7|     |14: X18
                    X25 : 8|     |13: X16              X27 : 8|     |13: X2
                     XD : 9|     |12: INIT              XD : 9|     |12: INIT
                     E3 :10|     |11: X9               X11 :10|     |11: D0
                        '--+-u--u+--'                      '--+-u--u+--'

*** Logic Map - 32 BIT CRC CHECKER
 Gbl Inp .--.      I/O  .--+--A--+--.  I/O            I/O  .--+--B--+--.  I/O
     INIT| 0|           | 0| *   |21|                      | 0| .   |21|
       D3| 1|        X8 | 1| 8   |20|                      | 1| *   |20|
   CLOCK2| 2|           | 2| .   |19|                  X26 | 2| 4   |19|
         | 3|        E1 | 3| 1   |18|                  X10 | 3| 8   |18|
         | 4|           | 4| .   |17|                  X27 | 4| 2   |17|
   CLOCK1| 5|       X24 | 5| 4   |16|                   XD | 5| 2   |16|
         '--'           | 6| .  4|15| X9               X18 | 6| 2  .|15|
                    X25 | 7| 4  2|14| X1                   | 7| .  4|14| X3
                    X16 | 8| 2  .|13|                   E2 | 8| 1  8|13| X11
                        | 9| .  1|12| X0                   | 9| .  *|12|
                    X17 |10| 2  .|11|                  X19 |10| 2  4|11| X2
                        '--+--C--+--'                      '--+--D--+--'
                        | 0| *   |21|                   X7 | 0| 8   |21|
                     X5 | 1| 8   |20|                  X23 | 1| 4   |20|
                        | 2| .   |19|                      | 2| *   |19|
                     E3 | 3| 1   |18|                  X14 | 3| 4   |18|
                 ERRFLG | 4| 1   |17|                  X15 | 4| 2   |17|
                    X12 | 5| 8   |16|                   X6 | 5| 4   |16|
                        | 6| *  .|15|                   XA | 6| 2  2|15| X22
                    X13 | 7| 8  1|14| X21               E4 | 7| 1  .|14|
                    X28 | 8| 2  1|13| X20               XB | 8| 2  .|13|
                     X4 | 9| 4  2|12| X29               XC | 9| 2  .|12|
                        |10| *  .|11|                  X30 |10| 1  1|11| X31
                        '--+-u--u+--'                      '--+-u--u+--'
```

```
*** Pin Map - 32 BIT CRC CHECKER

                                   .  |        .
                                .  |  |        |  .
                             .  |  |  |        |  |  .
                          .  |  |  |  |        |  |  |  .
                          |  |  |  |  |        |  |  |  |
                    .-----'--'--'--'--'--o-----'--'--'--'---.
                    |                       4  4  4  4  4   |
                    |     6  5  4  3  2  1  4  3  2  1  0   |
                    | 7                                   39|D0
                    | 8                  G  V             38|D1
                    | 9                  n  c             37|
                INIT|10                  d  c             36|D2
                  D3|11                                   35|CLOCK1
                Gnd |12            MACH-210               34| Gnd
              CLOCK2|13                                   33|
                    |14             V  G                  32|
                    |15             c  n                  31|
                    |16             c  d                  30|
                    |17                                   29|
                    |   1  1  2  2  2  2  2  2  2  2  2     |
                    |   8  9  0  1  2  3  4  5  6  7  8     |
                    '---.--.--.--.--------.--.--.--.--.-----'
                        |  |  |  |        |  |  |  |  |
                        '  |  |  |        |  |  |  |  '
                           '  |  |        |  |  |  '
                              '  |        |  |  ERRFLG
                                 '        |  '
                                          |
                                          '

The Design Doc is  stored in ===> X1.Rpt
The Jedec Data is  stored in ===> X1.Jed
The Placements are stored in ===> X1.Plc

%% FITR %% Error Count: 0, Warning Count: 0
%% FITR %% File Processed Successfully. - File: X1.pds
```

PLACEMENT FILE: MACH210 CHECKER

```
; Flags Used:          Unplace=False           Max Packing=False
; Flags Used:        Expand Small=True          Expand All=True

; FITR generated placements
Pin    10          INIT  ; Inp         ; I  0
Pin    11            D3  ; Inp         ; I  1
Pin    13        CLOCK2  ; Inp         ; I  2
Pin    26        ERRFLG    Reg         ; C  4
Pin    35        CLOCK1  ; Inp         ; I  5
Pin    36            D2  ; Inp         ; D 14
Pin    38            D1  ; Inp         ; D 10
Pin    39            D0  ; Inp         ; D  8
Node    3            X8    Reg         ; A  1
Node    5            E1  Comb          ; A  3
Node    7           X24    Reg         ; A  5
Node    9           X25    Reg         ; A  7
Node   10           X16    Reg         ; A  8
Node   12           X17    Reg         ; A 10
Node   14            X0    Reg         ; A 12
Node   16            X1    Reg         ; A 14
Node   17            X9    Reg         ; A 15
Node   20           X26    Reg         ; B  2
Node   21           X10    Reg         ; B  3
Node   22           X27    Reg         ; B  4
Node   23            XD    Reg         ; B  5
Node   24           X18    Reg         ; B  6
Node   26            E2  Comb          ; B  8
Node   28           X19    Reg         ; B 10
Node   29            X2    Reg         ; B 11
Node   31           X11    Reg         ; B 13
Node   32            X3    Reg         ; B 14
Node   35            X5    Reg         ; C  1
Node   37            E3  Comb          ; C  3
Node   39           X12    Reg         ; C  5
Node   41           X13    Reg         ; C  7
Node   42           X28    Reg         ; C  8
Node   43            X4    Reg         ; C  9
Node   46           X29    Reg         ; C 12
Node   47           X20    Reg         ; C 13
Node   48           X21    Reg         ; C 14
Node   50            X7    Reg         ; D  0
Node   51           X23    Reg         ; D  1
Node   53           X14    Reg         ; D  3
Node   54           X15    Reg         ; D  4
Node   55            X6    Reg         ; D  5
Node   56            XA    Reg         ; D  6
Node   57            E4  Comb          ; D  7
Node   58            XB    Reg         ; D  8
Node   59            XC    Reg         ; D  9
Node   60           X30    Reg         ; D 10
Node   61           X31    Reg         ; D 11
Node   65           X22    Reg         ; D 15
```

DESIGN FILE: MACH230 GENERATOR/CHECKER

```
TITLE     4 BIT PARALLEL IN - 32 BIT CRC GENERATOR/CHECKER
PATTERN   FINAL
REVISION  A
AUTHOR    PHIL MADISON/GENE CROSSLEY
COMPANY   AT&T PARADYNE/ADVANCED MICRO DEVICES
DATE      11-21-90, 12-04-91
CHIP      CRCCHK          MACH230
PIN 65 CLOCK0
PIN 62 CLOCK1 ;
PIN 23 CLOCK2 ;
PIN 20 CLOCK3
PIN 3 ERRFLG REGISTERED ;
PIN 41 IN0 ;
PIN 82 IN1 ;
PIN 83 INIT ;
PIN 61 OUT0 COMBINATORIAL ;
PIN 60 OUT1 COMBINATORIAL ;
PIN 45 OUT2 COMBINATORIAL ;
PIN 59 OUT3 COMBINATORIAL ;
PIN 24 OUT4 COMBINATORIAL ;
PIN 25 OUT5 COMBINATORIAL ;
PIN 26 OUT6 COMBINATORIAL ;
PIN 40 OUT7 COMBINATORIAL ;
PIN 76 RD0 ;
PIN 80 RD1 ;
PIN 77 RD2 ;
PIN 81 RD3 ;
PIN 79 TD0 ;
PIN 78 TD1 ;
PIN 75 TD2 ;
PIN 66 TD3 ;
NODE 12 E1 COMBINATORIAL ;
NODE 20 E2 COMBINATORIAL ;
NODE 125 E3 COMBINATORIAL ;
NODE 104 E4 COMBINATORIAL ;
NODE 10 RX0 REGISTERED ;
NODE 11 RX1 REGISTERED ;
NODE 29 RX2 REGISTERED ;
NODE 22 RX3 REGISTERED ;
NODE 121 RX4 REGISTERED ;
NODE 115 RX5 REGISTERED ;
NODE 101 RX6 REGISTERED ;
NODE 99 RX7 REGISTERED ;
NODE 3 RX8 REGISTERED ;
NODE 4 RX9 REGISTERED ;
NODE 19 RX10 REGISTERED ;
NODE 21 RX11 REGISTERED ;
NODE 117 RX12 REGISTERED ;
NODE 120 RX13 REGISTERED ;
NODE 109 RX14 REGISTERED ;
NODE 110 RX15 REGISTERED ;
NODE 8 RX16 REGISTERED ;
NODE 9 RX17 REGISTERED ;
NODE 27 RX18 REGISTERED ;
NODE 28 RX19 REGISTERED ;
NODE 123 RX20 REGISTERED ;
NODE 124 RX21 REGISTERED ;
NODE 107 RX22 REGISTERED ;
NODE 108 RX23 REGISTERED ;
NODE 6 RX24 REGISTERED ;
NODE 7 RX25 REGISTERED ;
NODE 23 RX26 REGISTERED ;
```

```
NODE 26 RX27 REGISTERED ;
NODE 114 RX28 REGISTERED ;
NODE 116 RX29 REGISTERED ;
NODE 103 RX30 REGISTERED ;
NODE 105 RX31 REGISTERED ;
NODE 98 RXA REGISTERED ;
NODE 100 RXB REGISTERED ;
NODE 102 RXC REGISTERED ;
NODE 18 RXD REGISTERED ;
NODE 49 /TX0 REGISTERED ;
NODE 37 /TX1 REGISTERED ;
NODE 55 /TX2 REGISTERED ;
NODE 54 /TX3 REGISTERED ;
NODE 92 /TX4 REGISTERED ;
NODE 83 /TX5 REGISTERED ;
NODE 68 /TX6 REGISTERED ;
NODE 67 /TX7 REGISTERED ;
NODE 35 /TX8 REGISTERED ;
NODE 39 /TX9 REGISTERED ;
NODE 53 /TX10 REGISTERED ;
NODE 52 /TX11 REGISTERED ;
NODE 87 /TX12 REGISTERED ;
NODE 91 /TX13 REGISTERED ;
NODE 80 /TX14 REGISTERED ;
NODE 73 /TX15 REGISTERED ;
NODE 43 /TX16 REGISTERED ;
NODE 44 /TX17 REGISTERED ;
NODE 59 /TX18 REGISTERED ;
NODE 60 /TX19 REGISTERED ;
NODE 94 /TX20 REGISTERED ;
NODE 95 /TX21 REGISTERED ;
NODE 78 /TX22 REGISTERED ;
NODE 71 /TX23 REGISTERED ;
NODE 41 /TX24 REGISTERED ;
NODE 42 /TX25 REGISTERED ;
NODE 57 /TX26 REGISTERED ;
NODE 58 /TX27 REGISTERED ;
NODE 88 /TX28 REGISTERED ;
NODE 90 /TX29 REGISTERED ;
NODE 76 /TX30 REGISTERED ;
NODE 69 /TX31 REGISTERED ;
NODE 70 TXA REGISTERED ;
NODE 72 TXB REGISTERED ;
NODE 74 TXC REGISTERED ;
NODE 56 TXD REGISTERED ;

GROUP MACH_SEG_A RX8 RX24 RX25 RX16 RX17 RX0 RX1 RX9 E1
GROUP MACH_SEG_B RX26 RX10 RX27 RXD RX18 RX19 RX2 RX11 RX3 E2
GROUP MACH_SEG_H RX5 RX12 RX13 RX28 RX4 RX29 RX20 RX21 E3
GROUP MACH_SEG_G RX7 RX23 RX14 RX15 RX6 RXA RXB RXC RX30 RX31 RX22 E4
GROUP MACH_SEG_C OUT4 TX8 OUT5 OUT6 TX24 TX25 TX16 TX17 TX0 TX1 TX9
GROUP MACH_SEG_D OUT7 TX26 TX10 TX27 TXD TX18 TX19 TX2 TX11 TX3
GROUP MACH_SEG_F OUT0 TX5 OUT1 TX12 OUT3 TX13 TX28 TX4 TX29 TX20 TX21
GROUP MACH_SEG_E TX7 TX23 TX14 TX15 TX6 TXA TXB TXC TX30 TX31 OUT2
TX22

EQUATIONS

RXA      := RD0:+:RX31
RXA.CLKF = CLOCK2
RXA.RSTF = GND
RXA.SETF = INIT

RXB      := RD1:+:RX30
RXB.CLKF = CLOCK2
```

```
RXB.RSTF = GND
RXB.SETF = INIT

RXC      := RD2:+:RX29
RXC.CLKF = CLOCK2
RXC.RSTF = GND
RXC.SETF = INIT

RXD      := RD3:+:RX28
RXD.CLKF = CLOCK2
RXD.RSTF = GND
RXD.SETF = INIT

RX0      := RXD
RX0.CLKF = CLOCK1
RX0.RSTF = GND
RX0.SETF = INIT

RX1      := RXD:+:RXC
RX1.CLKF = CLOCK1
RX1.RSTF = GND
RX1.SETF = INIT

RX2      := RXD:+:RXC:+:RXB
RX2.CLKF = CLOCK1
RX2.RSTF = GND
RX2.SETF = INIT

RX3      := RXC:+:RXB:+:RXA
RX3.CLKF = CLOCK1
RX3.RSTF = GND
RX3.SETF = INIT

RX4      := RXD:+:RXB:+:RXA
RX4.CLKF = CLOCK1
RX4.RSTF = GND
RX4.SETF = INIT

RX5      := RXD:+:RXA:+:RXC:+:RX1
RX5.CLKF = CLOCK1
RX5.RSTF = GND
RX5.SETF = INIT

RX6      := RXC:+:RXB:+:RX2
RX6.CLKF = CLOCK1
RX6.RSTF = GND
RX6.SETF = INIT

RX7      := RXD:+:RXB:+:RXA:+:RX3
RX7.CLKF = CLOCK1
RX7.RSTF = GND
RX7.SETF = INIT

RX8      := RXD:+:RXC:+:RXA:+:RX4
RX8.CLKF = CLOCK1
RX8.RSTF = GND
RX8.SETF = INIT

RX9      := RXC:+:RXB:+:RX5
RX9.CLKF = CLOCK1
RX9.RSTF = GND
RX9.SETF = INIT

RX10      := RXD:+:RXB:+:RXA:+:RX6
RX10.CLKF = CLOCK1
RX10.RSTF = GND
```

```
RX10.SETF = INIT

RX11      := RXC:+:RXA:+:RX7:+:RXD
RX11.CLKF = CLOCK1
RX11.RSTF = GND
RX11.SETF = INIT

RX12      := RXD:+:RXB:+:RX8:+:RXC
RX12.CLKF = CLOCK1
RX12.RSTF = GND
RX12.SETF = INIT

RX13      := RXC:+:RXA:+:RX9:+:RXB
RX13.CLKF = CLOCK1
RX13.RSTF = GND
RX13.SETF = INIT

RX14      := RXB:+:RX10:+:RXA
RX14.CLKF = CLOCK1
RX14.RSTF = GND
RX14.SETF = INIT

RX15      := RXA:+:RX11
RX15.CLKF = CLOCK1
RX15.RSTF = GND
RX15.SETF = INIT

RX16      := RXD:+:RX12
RX16.CLKF = CLOCK1
RX16.RSTF = GND
RX16.SETF = INIT

RX17      := RXC:+:RX13
RX17.CLKF = CLOCK1
RX17.RSTF = GND
RX17.SETF = INIT

RX18      := RXB:+:RX14
RX18.CLKF = CLOCK1
RX18.RSTF = GND
RX18.SETF = INIT

RX19      := RXA:+:RX15
RX19.CLKF = CLOCK1
RX19.RSTF = GND
RX19.SETF = INIT

RX20      := RX16;
RX20.CLKF = CLOCK1
RX20.RSTF = GND
RX20.SETF = INIT

RX21      := RX17;
RX21.CLKF = CLOCK1
RX21.RSTF = GND
RX21.SETF = INIT

RX22      := RXD:+:RX18
RX22.CLKF = CLOCK1
RX22.RSTF = GND
RX22.SETF = INIT

RX23      := RXD:+:RXC:+:RX19
RX23.CLKF = CLOCK1
```

```
RX23.RSTF = GND
RX23.SETF = INIT

RX24      := RXC:+:RXB:+:RX20
RX24.CLKF = CLOCK1
RX24.RSTF = GND
RX24.SETF = INIT

RX25      := RXB:+:RXA:+:RX21
RX25.CLKF = CLOCK1
RX25.RSTF = GND
RX25.SETF = INIT

RX26      := RXD:+:RXA:+:RX22
RX26.CLKF = CLOCK1
RX26.RSTF = GND
RX26.SETF = INIT

RX27      := RXC:+:RX23
RX27.CLKF = CLOCK1
RX27.RSTF = GND
RX27.SETF = INIT

RX28      := RXB:+:RX24
RX28.CLKF = CLOCK1
RX28.RSTF = GND
RX28.SETF = INIT

RX29      := RXA:+:RX25
RX29.CLKF = CLOCK1
RX29.RSTF = GND
RX29.SETF = INIT

RX30      := RX26
RX30.CLKF = CLOCK1
RX30.RSTF = GND
RX30.SETF = INIT

RX31      := RX27
RX31.CLKF = CLOCK1
RX31.RSTF = GND
RX31.SETF = INIT
E1 = RX5+/RX12+/RX13+RX4+/RX20+/RX21
E2 = RX28+/RX7+/RX23+RX14+RX15+RX6+/RX22
E3 = RX0+RX8+RX24+RX25+/RX16+/RX17+RX1+/RX9
E4 = RX26+RX10+/RX27+RX18+/RX19+/RX2+RX11+RX3+/RX29+RX30+RX31
ERRFLG      := E1+E2+E3+E4
ERRFLG.CLKF = CLOCK2
ERRFLG.RSTF = GND
ERRFLG.SETF = INIT
TXA      := TD0:+:TX31
TXA.CLKF = CLOCK0
TXA.RSTF = INIT
TXA.SETF = GND
TXB      := TD1:+:TX30
TXB.CLKF = CLOCK0
TXB.RSTF = INIT
TXB.SETF = GND
TXC      := TD2:+:TX29
TXC.CLKF = CLOCK0
TXC.RSTF = INIT
TXC.SETF = GND
```

```
TXD       := TD3:+:TX28
TXD.CLKF = CLOCK0
TXD.RSTF = INIT
TXD.SETF = GND
TX0       := TXD
TX0.CLKF = CLOCK3
TX0.RSTF = INIT
TX0.SETF = GND

TX1       := TXD:+:TXC
TX1.CLKF = CLOCK3
TX1.RSTF = INIT
TX1.SETF = GND

TX2       := TXD:+:TXC:+:TXB
TX2.CLKF = CLOCK3
TX2.RSTF = INIT
TX2.SETF = GND

TX3       := TXC:+:TXB:+:TXA
TX3.CLKF = CLOCK3
TX3.RSTF = INIT
TX3.SETF = GND

TX4       := TXD:+:TXB:+:TXA
TX4.CLKF = CLOCK3
TX4.RSTF = INIT
TX4.SETF = GND

TX5       := TXD:+:TXA:+:TXC:+:TX1
TX5.CLKF = CLOCK3
TX5.RSTF = INIT
TX5.SETF = GND

TX6       := TXC:+:TXB:+:TX2
TX6.CLKF = CLOCK3
TX6.RSTF = INIT
TX6.SETF = GND

TX7       := TXD:+:TXB:+:TXA:+:TX3
TX7.CLKF = CLOCK3
TX7.RSTF = INIT
TX7.SETF = GND

TX8       := TXD:+:TXC:+:TXA:+:TX4
TX8.CLKF = CLOCK3
TX8.RSTF = INIT
TX8.SETF = GND

TX9       := TXC:+:TXB:+:TX5
TX9.CLKF = CLOCK3
TX9.RSTF = INIT
TX9.SETF = GND

TX10       := TXD:+:TXB:+:TXA:+:TX6
TX10.CLKF = CLOCK3
TX10.RSTF = INIT
TX10.SETF = GND

TX11       := TXC:+:TXA:+:TX7:+:TXD
TX11.CLKF = CLOCK3
TX11.RSTF = INIT
TX11.SETF = GND
```

```
TX12      := TXD:+:TXB:+:TX8:+:TXC
TX12.CLKF = CLOCK3
TX12.RSTF = INIT
TX12.SETF = GND

TX13      := TXC:+:TXA:+:TX9:+:TXB
TX13.CLKF = CLOCK3
TX13.RSTF = INIT
TX13.SETF = GND

TX14      := TXB:+:TX10:+:TXA
TX14.CLKF = CLOCK3
TX14.RSTF = INIT
TX14.SETF = GND

TX15      := TXA:+:TX11
TX15.CLKF = CLOCK3
TX15.RSTF = INIT
TX15.SETF = GND

TX16      := TXD:+:TX12
TX16.CLKF = CLOCK3
TX16.RSTF = INIT
TX16.SETF = GND

TX17      := TXC:+:TX13
TX17.CLKF = CLOCK3
TX17.RSTF = INIT
TX17.SETF = GND

TX18      := TXB:+:TX14
TX18.CLKF = CLOCK3
TX18.RSTF = INIT
TX18.SETF = GND

TX19      := TXA:+:TX15
TX19.CLKF = CLOCK3
TX19.RSTF = INIT
TX19.SETF = GND

TX20      := TX16;
TX20.CLKF = CLOCK3
TX20.RSTF = INIT
TX20.SETF = GND

TX21      := TX17;
TX21.CLKF = CLOCK3
TX21.RSTF = INIT
TX21.SETF = GND

TX22      := TXD:+:TX18
TX22.CLKF = CLOCK3
TX22.RSTF = INIT
TX22.SETF = GND

TX23      := TXD:+:TXC:+:TX19
TX23.CLKF = CLOCK3
TX23.RSTF = INIT
TX23.SETF = GND

TX24      := TXC:+:TXB:+:TX20
TX24.CLKF = CLOCK3
TX24.RSTF = INIT
```

```
TX24.SETF = GND

TX25      := TXB:+:TXA:+:TX21
TX25.CLKF = CLOCK3
TX25.RSTF = INIT
TX25.SETF = GND

TX26      := TXD:+:TXA:+:TX22
TX26.CLKF = CLOCK3
TX26.RSTF = INIT
TX26.SETF = GND

TX27      := TXC:+:TX23
TX27.CLKF = CLOCK3
TX27.RSTF = INIT
TX27.SETF = GND

TX28      := TXB:+:TX24
TX28.CLKF = CLOCK3
TX28.RSTF = INIT
TX28.SETF = GND

TX29      := TXA:+:TX25
TX29.CLKF = CLOCK3
TX29.RSTF = INIT
TX29.SETF = GND

TX30      := TX26
TX30.CLKF = CLOCK3
TX30.RSTF = INIT
TX30.SETF = GND

TX31      := TX27
TX31.CLKF = CLOCK3
TX31.RSTF = INIT
TX31.SETF = GND
OUT0 = /IN0*/IN1*TX0 +
       /IN0*IN1*TX8  +
        IN0*/IN1*TX16 +
        IN0*IN1*TX24
OUT1 = /IN0*/IN1*TX1 +
       /IN0*IN1*TX9  +
        IN0*/IN1*TX17 +
        IN0*IN1*TX25
OUT2 = /IN0*/IN1*TX2 +
       /IN0*IN1*TX10 +
        IN0*/IN1*TX18 +
        IN0*IN1*TX26
OUT3 = /IN0*/IN1*TX3 +
       /IN0*IN1*TX11 +
        IN0*/IN1*TX19 +
        IN0*IN1*TX27
OUT4 = /IN0*/IN1*TX4 +
       /IN0*IN1*TX12 +
        IN0*/IN1*TX20 +
        IN0*IN1*TX28
OUT5 = /IN0*/IN1*TX5 +
       /IN0*IN1*TX13 +
        IN0*/IN1*TX21 +
        IN0*IN1*TX29
```

```
OUT6 = /IN0*/IN1*TX6 +
        /IN0*IN1*TX14 +
         IN0*/IN1*TX22 +
         IN0*IN1*TX30
OUT7 = /IN0*/IN1*TX7 +
        /IN0*IN1*TX15 +
         IN0*/IN1*TX23 +
         IN0*IN1*TX31
SIMULATION
```

FITTER FILE: MACH230 GENERATOR/CHECKER

Note: This file has been condensed in order to save trees.

```
PALASM 4 v1.3 MACH FITR - MARKET RELEASE (12-12-91)
 (C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990,1991
************************************************************************************
     Mach PLD Fitter - v 1.60          4 BIT PARALLEL IN - 32 BIT CRC GENERATOR/
CHECKER
************************************************************************************
*** Timing Analysis for Signals
 Parameter    Min    Max          Signal List (Those having Max delay.)
      Tpd      1      1              OUT7             OUT6             OUT5
                                     OUT4             OUT3             OUT2
                                     OUT1             OUT0
      Tsu      1      1               TXD              TXC              TXB
                                      TXA             TX31             TX30
                                     TX29              RXD
      Tco      0      1                E3               E4               E1
                                       E2             OUT0             OUT1
                                     OUT2             OUT7
      Tcr      1      2            ERRFLG
.
 Key:
 Tpd - Combinatorial propagation delay, input to output
 Tsu - Combinatorial setup delay before clock
 Tco - clock to output (register output to combinatorial output)
 Tcr - Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
 All delay values are expressed in terms of array passes
*** Device Resource Checks
                Available          Used          Remaining
        Clocks:     4                4               0
          Pins:    70               24              46       ->       34%
     I/O Macro:    64                9              55
   Total Macro:   128               85              43
 Product Terms:   512              295             116       ->       76%

MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
*** Last Equations Placed in Blocks
Weakly - |> WARNING F176 - MACH 230 Block Reletter
|> INFORMATION F050 - Device Utilization....... *: 57 %
```

```
*** Feedback Map - 4 BIT PARALLEL IN - 32 BIT CRC GENERATOR/CHECKER
Gbl Inp .--.     I/O  .--+--A--+--.  I/O            I/O  .--+--B--+--.  I/O
        | 0|       E4 : 0|     |25| RX5              RX6 : 0|     |25|
        | 1|       E2 : 1|     |24|                      | 1|     |24| RD3
        | 2|      RX4 : 2|     |23|                      | 2|     |23|
        | 3|     RX20 : 3|     |22| E1               RXC : 3|     |22|
        | 4|     RX21 : 4|     |21|                 RX14 : 4|     |21| RX22
        | 5|     INIT : 5|     |20|                 INIT : 5|     |20|
        '--'          | 6|     |19|                      | 6|     |19|
                  RXA : 7|     |18| RX12             RXA : 7|     |18| RX28
                  RXB : 8|     |17|                  RXB : 8|     |17|
                  RXC : 9|     |16| RXD                  | 9|     |16| RX7
                      |10|     |15|                  RXD :10|     |15|
                 RX13 :11|     |14|                      |11|     |14|
                      |12|     |13: E3              RX23 :12|     |13: RX15
                      '--+--C--+--'                      '--+--D--+--'
                  TXA : 0|     |25| IN1              TXA : 0|     |25| IN1
                  TXB : 1|     |24|                  TXB : 1|     |24|
                /TX28 : 2|     |23|                /TX15 : 2|     |23|
                /TX13 : 3|     |22| IN0                  | 3|     |22| IN0
                 /TX6 : 4|     |21|                 /TX6 : 4|     |21|
                /TX21 : 5|     |20|                      | 5|     |20|
                  TXD : 6|     |19| /TX12            TXD : 6|     |19| /TX23
                 /TX5 : 7|     |18| /TX20            TXC : 7|     |18| /TX31
                /TX30 : 8|     |17| /TX4                 | 8|     |17|
                /TX22 : 9|     |16| /TX29          /TX22 : 9|     |16| /TX7
                /TX14 :10|     |15: TXC            /TX14 :10|     |15: /TX28
                      |11|     |14|                      |11|     |14: TD3
                      |12|     |13: INIT                 |12|     |13: INIT
                      '--+--E--+--'                      '--+--F--+--'
                  TXA : 0|     |25| IN1              TXA : 0|     |25| IN1
                  TXB : 1|     |24| TD2              TXB : 1|     |24| /TX0
                /TX26 : 2|     |23|                      | 2|     |23|
                /TX18 : 3|     |22| IN0            /TX16 : 3|     |22| IN0
                /TX11 : 4|     |21|                /TX11 : 4|     |21|
                 /TX3 : 5|     |20| TD1             /TX3 : 5|     |20| /TX17
                  TXD : 6|     |19| TD0              TXD : 6|     |19| /TX25
                  TXC : 7|     |18| /TX10           /TX8 : 7|     |18|
                /TX30 : 8|     |17|                      | 8|     |17|
                /TX31 : 9|     |16| /TX29           /TX1 : 9|     |16|
                 /TX2 :10|     |15|                 /TX9 :10|     |15: TXC
                      |11|     |14: INIT           /TX24 :11|     |14: INIT
                /TX27 :12|     |13: /TX19          /TX27 :12|     |13: /TX19
                      '--+--G--+--'                      '--+--H--+--'
                  RXD : 0|     |25|                  RXD : 0|     |25|
                 RX30 : 1|     |24|                 RX16 : 1|     |24|
                 RX31 : 2|     |23|                 RX17 : 2|     |23|
                  RXC : 3|     |22|                  RX1 : 3|     |22|
                  RX2 : 4|     |21| RD0                  | 4|     |21|
                 INIT : 5|     |20| RD2             INIT : 5|     |20|
                      | 6|     |19| RX29                 | 6|     |19| RX25
                  RXA : 7|     |18| RX3              RXA : 7|     |18|
                  RXB : 8|     |17| RD1              RXB : 8|     |17|
                 RX11 : 9|     |16| RX10             RXC : 9|     |16| RX8
                 RX26 :10|     |15|                      |10|     |15: RX0
                 RX27 :11|     |14|                      |11|     |14|
                 RX18 :12|     |13: RX19             RX9 :12|     |13: RX24
                      '--+-u--u+--'                      '--+-u--u+--'
```

```
*** Logic Map - 4 BIT PARALLEL IN - 32 BIT CRC GENERATOR/CHECKER

 Gbl Inp .--.       I/O .--+--A--+--.  I/O           I/O .--+--B--+--.  I/O
   CLOCK3| 0|    ERRFLG | 0| 1   |25|                RXD | 0| 2   |25|
   CLOCK2| 1|       RX8 | 1| 8   |24|               RX10 | 1| 8   |24|
      IN0| 2|       RX9 | 2| 4   |23|                 E2 | 2| 1   |23|
   CLOCK1| 3|           | 3| *   |22|               RX11 | 3| 8   |22|
   CLOCK0| 4|      RX24 | 4| 4   |21|                RX3 | 4| 4   |21|
     INIT| 5|      RX25 | 5| 4   |20|               RX26 | 5| 4   |20|
         '--'      RX16 | 6| 2   |19|                    | 6| *   |19|
                   RX17 | 7| 2   |18|                    | 7| *   |18|
                    RX0 | 8| 1   |17|               RX27 |-8| 2   |17|
                    RX1 | 9| 2   |16|               RX18 | 9| 2   |16|
                     E1 |10| 1  .|15|               RX19 |10| 2  .|15|
                        |11| .  .|14|                RX2 |11| 4  .|14|
                        |12| .  .|13|                    |12| .  .|13|
                        '--+--C--+--'                    '--+--D--+--'
                   OUT4 | 0| 4   |25|               OUT7 | 0| 4   |25|
                   /TX8 | 1| 8   |24|                    | 1| *   |24|
                   OUT5 | 2| 4   |23|              /TX11 | 2| 8   |23|
                   /TX1 | 3| 2   |22|              /TX10 | 3| 8   |22|
                   OUT6 | 4| 4   |21|               /TX3 | 4| 4   |21|
                   /TX9 | 5| 4   |20|               /TX2 | 5| 4   |20|
                        | 6| *   |19|                TXD | 6| 2   |19|
                  /TX24 | 7| 4   |18|              /TX26 | 7| 4   |18|
                  /TX25 | 8| 4   |17|              /TX27 | 8| 2   |17|
                  /TX16 | 9| 2   |16|              /TX18 | 9| 2   |16|
                  /TX17 |10| 2  1|15|  /TX0        /TX19 |10| 2  .|15|
                        |11| .  .|14|                    |11| *  .|14|
                        |12| .  .|13|                    |12| .  .|13|
                        '--+--E--+--'                    '--+--F--+--'
                   OUT2 | 0| 4   |25|               OUT0 | 0| 4   |25|
                   /TX7 | 1| 8   |24|               /TX5 | 1| 8   |24|
                   /TX6 | 2| 4   |23|               OUT1 | 2| 4   |23|
                  /TX31 | 3| 1   |22|                    | 3| *   |22|
                    TXA | 4| 2   |21|               OUT3 | 4| 4   |21|
                  /TX23 | 5| 4   |20|              /TX12 | 5| 8   |20|
                    TXB | 6| 2   |19|              /TX28 | 6| 2   |19|
                  /TX15 | 7| 2   |18|                    | 7| *   |18|
                    TXC | 8| 2   |17|              /TX29 | 8| 2   |17|
                        | 9| *   |16|              /TX13 | 9| 8   |16|
                  /TX30 |10| 1  .|15|               /TX4 |10| 4  .|15|
                        |11| .  4|14|  /TX14             |11| *  .|14|
                  /TX22 |12| 2  .|13|              /TX20 |12| 1  1|13|  /TX21
                        '--+--G--+--'                    '--+--H--+--'
                    RXA | 0| 2   |25|               RX28 | 0| 2   |25|
                    RX7 | 1| 8   |24|                RX5 | 1| 8   |24|
                    RXB | 2| 2   |23|               RX29 | 2| 2   |23|
                    RX6 | 3| 4   |22|               RX12 | 3| 8   |22|
                    RXC | 4| 2   |21|                    | 4| *   |21|
                   RX30 | 5| 1   |20|                    | 5| *   |20|
                     E4 | 6| 1   |19|               RX13 | 6| 8   |19|
                   RX31 | 7| 1   |18|                RX4 | 7| 4   |18|
                        | 8| *   |17|                    | 8| *   |17|
                   RX22 | 9| 2   |16|               RX20 | 9| 1   |16|
                   RX23 |10| 4  .|15|               RX21 |10| 1  .|15|
                   RX14 |11| 4  .|14|                 E3 |11| 1  .|14|
                   RX15 |12| 2  .|13|                    |12| .  .|13|
                        '--+-u--u+--'                    '--+-u--u+--'
```

```
*** Pin Map - 4 BIT PARALLEL IN - 32 BIT CRC GENERATOR/CHECKER


                                            INIT
                               ERRFLG       | IN1
                                  . |       | | RD3
                                . | |       | | | RD1
                              . | | |       | | | | TD0
                            . | | | |       | | | | | TD1
                          . | | | | |       | | | | | | RD2
                        . | | | | | |       | | | | | | | RD0
                      . | | | | | | |       | | | | | | | | TD2
                      | | | | | | | |       | | | | | | | | |
               .------'-'-'-'-'-'-'-'---o---'-'-'-'-'-'-'-'-'----.
               |    1 1                   8 8 8 8 8 7 7 7 7 7    |
               |    1 0 9 8 7 6 5 4 3 2 1 4 3 2 1 0 9 8 7 6 5    |
               |12                                         Gnd 74|
               |13  G                 V G V                    73|
               |14  n                 c n c                    72|
               |15  d                 c d c                    71|
               |16                                             70|
               |17                                             69|
               |18                                             68|
               |19                                             67|
         CLOCK3|20                                             66|TD3
               |21 Vcc               MACH-230                  65|CLOCK0
               |22 Gnd                                     Gnd 64|
         CLOCK2|23                                         Vcc 63|
           OUT4|24                                             62|CLOCK1
           OUT5|25                                             61|OUT0
           OUT6|26                                             60|OUT1
               |27                                             59|OUT3
               |28                                             58|
               |29                    V G V                 G  57|
               |30                    c n c                 n  56|
               |31                    c d c                 d  55|
               |32 Gnd                                         54|
               |    3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5    |
               |    3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3    |
               '----.-.-.-.-.-.-.-.-.-------.-.-.-.-.-.-.-.------'
                    | | | | | | | | |       | | | | | | | |
                    ' | | | | | | | |       | | | | | | | |
                      ' | | | | | | |       | | | | | | | '
                        ' | | | | | |       | | | | | | '
                          ' | | | | |       | | | | | '
                            ' | | | |       | | | | '
                              ' | | |       | | | '
                                ' | |       | | '
                               OUT7 |       | '
                                  IN0       OUT2
The Design Doc is  stored in ===> Comba.Rpt
The Jedec Data is  stored in ===> Comba.Jed
The Placements are stored in ===> Comba.Plc

%% FITR %% Error Count: 0, Warning Count: 1
%% FITR %% File Processed Successfully. - File: Comba.pds
```

PLACEMENT FILE: MACH230 GENERATOR/CHECKER

```
Flags Used:            Unplace=False          Max Packing=True
Flags Used:       Expand Small=False          Expand All=False
; FITR generated placements
Pin   65          CLOCK0  ; Inp          ; I  4
Pin   62          CLOCK1  ; Inp          ; I  3
Pin   23          CLOCK2  ; Inp          ; I  1
Pin   20          CLOCK3  ; Inp          ; I  0
Pin    3          ERRFLG    Reg          ; A  0
Pin   41             IN0  ; Inp          ; I  2
Pin   82             IN1  ; Inp          ; H  0
Pin   83            INIT  ; Inp          ; I  5
Pin   61            OUT0    Comb         ; F  0
Pin   60            OUT1    Comb         ; F  2
Pin   45            OUT2    Comb         ; E  0
Pin   59            OUT3    Comb         ; F  4
Pin   24            OUT4    Comb         ; C  0
Pin   25            OUT5    Comb         ; C  2
Pin   26            OUT6    Comb         ; C  4
Pin   40            OUT7    Comb         ; D  0
Pin   76             RD0  ; Inp          ; H 12
Pin   80             RD1  ; Inp          ; H  4
Pin   77             RD2  ; Inp          ; H 10
Pin   81             RD3  ; Inp          ; H  2
Pin   79             TD0  ; Inp          ; H  6
Pin   78             TD1  ; Inp          ; H  8
Pin   75             TD2  ; Inp          ; H 14
Pin   66             TD3  ; Inp          ; G  0
Node  12              E1    Comb         ; A 10
Node  20              E2    Comb         ; B  2
Node 125              E3    Comb         ; H 11
Node 104              E4    Comb         ; G  6
Node  10             RX0    Reg          ; A  8
Node  11             RX1    Reg          ; A  9
Node  29             RX2    Reg          ; B 11
Node  22             RX3    Reg          ; B  4
Node 121             RX4    Reg          ; H  7
Node 115             RX5    Reg          ; H  1
Node 101             RX6    Reg          ; G  3
Node  99             RX7    Reg          ; G  1
Node   3             RX8    Reg          ; A  1
Node   4             RX9    Reg          ; A  2
Node  19            RX10    Reg          ; B  1
Node  21            RX11    Reg          ; B  3
Node 117            RX12    Reg          ; H  3
Node 120            RX13    Reg          ; H  6
Node 109            RX14    Reg          ; G 11
Node 110            RX15    Reg          ; G 12
Node   8            RX16    Reg          ; A  6
Node   9            RX17    Reg          ; A  7
Node  27            RX18    Reg          ; B  9
Node  28            RX19    Reg          ; B 10
Node 123            RX20    Reg          ; H  9
Node 124            RX21    Reg          ; H 10
Node 107            RX22    Reg          ; G  9
Node 108            RX23    Reg          ; G 10
Node   6            RX24    Reg          ; A  4
Node   7            RX25    Reg          ; A  5
Node  23            RX26    Reg          ; B  5
Node  26            RX27    Reg          ; B  8
Node 114            RX28    Reg          ; H  0
Node 116            RX29    Reg          ; H  2
```

```
Node 103            RX30    Reg         ; G  5
Node 105            RX31    Reg         ; G  7
Node  98             RXA    Reg         ; G  0
Node 100             RXB    Reg         ; G  2
Node 102             RXC    Reg         ; G  4
Node  18             RXD    Reg         ; B  0
Node  49            /TX0    Reg         ; C 15
Node  37            /TX1    Reg         ; C  3
Node  55            /TX2    Reg         ; D  5
Node  54            /TX3    Reg         ; D  4
Node  92            /TX4    Reg         ; F 10
Node  83            /TX5    Reg         ; F  1
Node  68            /TX6    Reg         ; E  2
Node  67            /TX7    Reg         ; E  1
Node  35            /TX8    Reg         ; C  1
Node  39            /TX9    Reg         ; C  5
Node  53           /TX10    Reg         ; D  3
Node  52           /TX11    Reg         ; D  2
Node  87           /TX12    Reg         ; F  5
Node  91           /TX13    Reg         ; F  9
Node  80           /TX14    Reg         ; E 14
Node  73           /TX15    Reg         ; E  7
Node  43           /TX16    Reg         ; C  9
Node  44           /TX17    Reg         ; C 10
Node  59           /TX18    Reg         ; D  9
Node  60           /TX19    Reg         ; D 10
Node  94           /TX20    Reg         ; F 12
Node  95           /TX21    Reg         ; F 13
Node  78           /TX22    Reg         ; E 12
Node  71           /TX23    Reg         ; E  5
Node  41           /TX24    Reg         ; C  7
Node  42           /TX25    Reg         ; C  8
Node  57           /TX26    Reg         ; D  7
Node  58           /TX27    Reg         ; D  8
Node  88           /TX28    Reg         ; F  6
Node  90           /TX29    Reg         ; F  8
Node  76           /TX30    Reg         ; E 10
Node  69           /TX31    Reg         ; E  3
Node  70             TXA    Reg         ; E  4
Node  72             TXB    Reg         ; E  6
Node  74             TXC    Reg         ; E  8
Node  56             TXD    Reg         ; D  6
;
; Unused Pins & Nodes
;
; -> None Found.
;
```

Pixel Processor for a High-Speed Graphics Controller: MACH230

Advanced Micro Devices

Application Note

by Ray Broemmelsiek, Sony Corporation of America, United States

INTRODUCTION

By using the MACH230, a graphics controller design based on PAL22V10s and GAL6001s was migrated quickly. The Pixel Processor MACH-based design reduced parts count by 70, reduced board space by 31 square inches, and increased drawing performance by 25%. The flexibility of the MACH macrocell and switch matrix allowed 87% utilization of the part's resources.

One MACH230 contains the complete Pixel Processor. The Pixel Processor is a general-purpose BitBLT and drawing processor. It is used as a central component to the drawing engine in a high performance X-server. A single Pixel Processor supports one plane of the frame buffer. In this design, 8-bit pixels are used, which requires 8 Pixel Processors. An additional MACH230 is used as a Mixing Controller. Together these components allow arbitrary movement of pixels between buffers, graphics primitive rendering, logical operations (raster ops) and pixel mixing from multiple sources at speeds in excess of 200 million pixels per second (see Figure 9-10).

FROM HIGH PAL DENSITY TO HIGH DENSITY PALS

As is too often the case, the engineers on this project were given a design task and schedule that allowed time only for everything to go right. This paper is a discussion of how the family of MACH parts were chosen and applied to a very large PAL design. It then discusses the specific application of the MACH230 to one design conversion process. This design conversion, which was key to the success of the product design, is the Pixel Processor.

We began with an engineering prototype design that used about three hundred PAL's over two large boards. It was necessary for us to migrate a large part of the design from the first board onto the second board complete with schematics and PAL equations in about six weeks. Because the second board was already fully populated, we required a solution that would free up additional board space as well as provide a straightforward transition from the existing design.

Preliminary Evaluation of MACH Parts for use in the Existing Design

The engineering prototype design consisted largely of 22V10s, 20V8s and GAL6001s. This design was especially sensitive to switching noise because of the design partitioning between the two boards, the large number of busses and the large number of devices that were simultaneously switching data through these buses. We felt our approach should include a higher level of integration to reduce board noise and routing congestion, as well as free up board space.

After examining other integration and parts packaging solutions, we considered the AMD MACH solution to be the most reasonable approach because the MACH macro cell is a superset of the 22V10 with equivalent performance, the density of the parts moves a good deal of the routing and pins from the board onto the chip, and because the MACH family was proven with the MACH110 and MACH130. Furthermore, AMD had been making prototype availability and support for additions to this family as scheduled.

MACH Performance and Macrocell Evaluation

Performance evaluation was simple because of the consistent timing characteristics throughout the part. There are additional features of the MACH parts that provide timing advantages over conventional 22V10 macro cells (such as shorter set-up times from I/O pins to buried registers) but it wasn't necessary to consider these at this stage.

The Tools Used for Rapid Development

One of the most effective tools to evaluate several parameters together with "what-if" scenarios is the spreadsheet. We use the spreadsheet as a tool to evaluate board space, cost, parts resource, and performance parameter trade-offs. This design is no exception. We needed to be certain about the board space but we also required a degree of comfort in evaluating the MACH resources for the design conversions.

The second significant tool we used which is a must for such a high degree of programmable logic, is a PLD design tool which allows design and multiple part simulation using a high level design language. Because PALASM is really only intended as a base point for other tool designers, and because the MACH parts we chose were only just being released, we chose a hybrid solution. We required PALASM for the fitter program but we used TANGO PLD for our design language. (TANGO PLD now supports the MACH parts directly). This HDL-like language is a C language subset with signal extensions for the PLD design environment. Then

Publication# 17753 Rev. A Amendment/0
Issue Date: May 1993

we wrote a small C routine which translated the syntax of the TANGO PLD product term equations (one form of output file from the TANGO PLD compiler) into PALASM product term equations. PALASM could then use this form to perform the fit and lastly generate a JEDEC file. Thus we had a rapid way of designing, testing and verifying resources while making the conversion.

Evaluating the PAL-Based Design for Conversion to Mach Parts

Candidates for design conversion were groups of PAL's that shared a large number of common signals. Of these groups, those PAL's that had a low number of product terms were candidates for the more cost-effective MACH 1 family. Otherwise the MACH 2 family could be used. The speed of the MACH parts to be selected was straightforward in most cases. The PAL's from the engineering prototype design mandated the same speed for the MACH parts. Preliminary evaluation of the system design produced five groups of PAL's as candidates for conversion to the MACH design. Further work in the conversion is required beyond this point in order to verify the actual fit.

A Closer Look at the MACH Resources for the Pixel Processor Design Change

The Pixel Processor conversion is the largest and most complex of the design conversions. Because of this, additional detailed evaluation is necessary. Again, using a spreadsheet, a close approximation of the implementation used for the Pixel Processor was used. The three resources that were evaluated were pin count, product terms, and inter-block routing (block input signals).

Repartitioning the Design for a MACH Implementation

There is a dramatic difference in the number of parts used in the prototype design as compared to the MACH conversion – from 80 parts to 10. One important reason for this is that the design could be partitioned in a different way because the MACH part allowed a much higher level of integration. Each of the 16 Pixel Processors in the prototype design operated on an 8 bit pixel. But in the MACH version, each of 8 Pixel Processors operates on 16 different one-bit pixels. Again because of the higher level of integration in the MACH parts, only one 16-way, 8 bit barrel shifter is required rather than two.

Timing

Once again, using a spreadsheet demonstrated a few of the critical timing parameters that were considered in order to meet the system performance requirements. In this case, a 40 MHz system clock with MACH230-20's were used. However, it is also possible to use MACH230-15's with a 50 MHz clock. Refer to the Figure 9-3. Pixel Processor Timing Specification which illustrates the significant Pixel Processor timing parameters in the final design.

Making the Design Fit into the MACH230

A detailed resource evaluation suggested that the Pixel Processor design may fit into a MACH230. However, because of the high level of the MACH's resource usage, the fitter was not successful without intervention. Therefore, it was necessary to make trade-offs in placement in order to best use the routing resources. The first level of intervention was finding a sensible way to "block out" the design into the MACH part. First refer to the Pixel Processor Detail (Figure 9-4) and the Pixel Processor Barrel Shifter Detail Figure 9-5 in order to understand how the Pixel Processor is broken down to fit into the MACH230. From these diagrams, the blocking can be translated as seen in Figure 9-6. Fitting 4 bits of Data Functions into 2 MACH230 Blocks.

Repartitioning the Design to Reduce Routing Congestion

To place the function blocks of the Pixel Processor successfully, it's important to first understand the available resources of the MACH's macro blocks. Each block has the possibility to receive a maximum of 26 inputs. However, there is no guarantee that an arbitrary 26 signals may be able to route successfully into the block. Also, there is a higher penalty paid for bringing global signals into the block rather than local signals (from the block's own input pins or macro-cell outputs) or sibling block signals (outputs from the block directly opposite the block being considered).

Finding Symmetry to Simplify Routing

There is simplicity in symmetry. With a heavily bussed design like the Pixel Processor, it is even more important to find its symmetry as it relates to the MACH part. As can be seen in Figure 9-2, Pixel Processor Block Diagram, it's clear that there is a great deal of symmetry in the design itself. By paying heed to the few rules mentioned above the design breaks up well into the configuration shown in Figure 9-5.

Almost all data lines remain local or shared with the sibling block. The only data lines that require global routing resources are the output data lines from the second stage of the 16-bit barrel shifter. By again examining figure 5 Pixel Processor Barrel Shifter Detail, it's clear that the connection arrangement requires that data lines cannot be confined to one single group of 4 bits.

Redefining the Control Interface

In the first pass of the MACH Pixel Processor implementation, System Bus data (4 bits per Pixel Processor), Color data (1 bit per Pixel Processor), ALU function code (4 bits per Pixel Processor), and barrel shifter code (4 bits per Pixel Processor), had dedicated pins into the

Pixel Processor. However, because of pin resource limitations and routing congestion, it became necessary to share pins between these various types of data.

The decision as to which functions can share pins is a design dependent one. This must be determined based on which functions are mutually exclusive in normal operation of the system. If routing resources are also restricted, then the designer must also consider how these mutually exclusive functions map into the internal representation (within the MACH part) of the design so as to minimize global routing.

In the Pixel Processor design, the Color data and the System Bus access is never used at the same time. This is because the Color data is used only during rendering and the System Bus is only used when the system bus accesses the DRAM or VRAM buffers to read or write. Because of this mutual exclusion and because the design mapping lent itself well to this configuration, the System Bus data and Color data pins are shared. (This requires expanding the Color data to four bits outside of the Pixel Processor). By sharing these pins, the need to distribute one global signal to four blocks was removed.

The second decision that was made was to multiplex the ALU function code and the Barrel shift code into the Pixel Processor two bits at a time. Doing this removed two ALU function signals from being globally routed to four blocks and two shift code signals from being globally routed to the four sibling blocks. However, in exchange, a globally routed select signal was added.

Meeting Timing Requirements

As was determined in timing analysis spreadsheets, the MACH230 meets the timing requirements to get on and off chip. However, one particular aspect of the MACH implementation of the Pixel Processor should be mentioned here because this is the limiting timing constraint for its throughput.

The Pixel Processor is pipelined internally. Two clocks are required to pass the incoming data through the barrel shifter and a third is required to pass the data out of the ALU. Provided the function and shift codes are available at the proper times, these three clocks would determine the throughput of the Pixel Processor. However, when the Color data is used (which requires external data expanding), or the System bus data is used, the throughput is reduced. These data (they share the same pins) are forced into the Pixel Processor's pipeline by using the asynchronous SET or RESET signals on the second stage of the Barrel shifter. (Figure 9-5).

The configuration of the MACH230 and its timing are the reasons for this data path reducing the throughput. In the MACH230, when any one macro cell in a block is enabled for using SET or RESET, all macro cells within that block are enabled. In the Pixel Processor, this results in destroying data in the pipeline before it can be used unless the throughput is reduced. In terms of the MACH230's timing, the asynchronous SET and RESET are somewhat slower than the registered functions. So the choice is to use two clock periods or slow down the clock period.

The advantage to using the SET and RESET to force data into the pipeline is that this approach does not require another 2 product terms. In some designs, it's not necessary to use this approach if there are product terms remaining in the quadruplet of the macro cell. However, in the second stage of the Barrel shifter in this design, all four product terms are used and there are not enough remaining within a block to steal from adjacent macrocells for all four bits of the second stage. So this approach is a nice way to conserve product terms.

Fixing Nodes

The final dimension in fitting the design is fixing the nodes. This process is actually a natural fall-out of blocking out the design. Fixing nodes means assigning certain signals to particular pins or buried cells. It is not necessary to assign all signals to nodes when using this method. In fact, such an approach may result in a failure to fit the design. The purpose of fixing nodes is only to steer the fitter in the right direction.

OVERVIEW OF THE PIXEL PROCESSOR FUNCTIONS

Pixel Processor General Function

The Pixel Processor is a general purpose 2D graphics design implemented on a single IC. It is only limited in that it operates on 16 pixels at a time. However, the design may be modified to operate on less if required.

The number of Pixel Processors used in a display controller design is based on the number of bit planes that the designer wants the display controller to operate on simultaneously. For example, with 8 Pixel Processors, a design could operate on 16 8-bit pixels in one pass or 16 24-bit pixels in 3 passes.

The Pixel Processor architecture is based on the backing and forward (visible) buffer architecture in which data is BLT'ed (transferred) into the visible buffer (or frame buffer) as the basis of rendering primitives. Because it has a 16 way barrel shifter, it can move 16 pixels from an arbitrary source address to an arbitrary destination address without regard to pixel boundary limitations. In addition, the Pixel Processor performs "raster op's" or logical operations on the pixels as they pass through it.

In order to provide system support, the Pixel Processor has an independent bi-directional port for system bus read and write operations into the frame and backing buffers. The Pixel Processor also has an independent

16-bit port that passes out the data that has been read from the RAM ports. This port is used when the RAM data is to be the source for a graphic primitive template (such as a font glyph or line segment). The data becomes the WRITE MASK to the opposing RAM buffer (usually the VRAM).

A Summary of the pixel operations supported by the Pixel Processor are listed below:

- BITBLT's
- Primitive Rendering
- Raster Operations (Logical Pixels)
- Mixing
- Anti-aliasing
- Three Source Operands (X-Window Pixel operations)
- Stippling

Data Paths

The Pixel Processor has data paths for three independent functions. These functions are BLT'ing data from one frame buffer to another, independent system access to these frame buffers, and write mask output which is the data sourced from the buffer that is read during a BLT operation.

DRAM/VRAM Data Interface

The DRAM/VRAM interfaces are the primary data paths of the Pixel Processor. They consists of two 16 bit bi-directional data busses, one each for the VRAM data bus and the DRAM data bus. This data path is used to BLT pixels in either direction between the DRAM and VRAM and to perform logical operations on the data as it passes through.

A typical operation would be the following sequence:

DRAM->VRAM BLT with Logic Operation

1) Read data from the DRAM data bus into the Pixel Processor.
2) Perform first stage barrel shift on the DRAM data to align it for writing into the VRAM.
3) Pass the data from the first stage out to the WRITE ENABLE data bus interface.
4) Perform second stage barrel shift.
5) Read data from VRAM port and perform logical operation with this data and the data in the second stage of the barrel shifter.
6) Write the resultant data back out the VRAM port.

Another typical case is as follows:

DRAM->VRAM Primitive Rendering in One Color

1) Receive (read) data (primitive template) from the DRAM data bus into the Pixel Processor.
2) Perform first stage barrel shift on the DRAM data to align it for writing into the VRAM.
3) Pass the data from the first stage out to the WRITE ENABLE data bus interface. This is where the data is used to make a write enable template for the VRAM.
4) Second stage barrel shift is not performed at this point. Instead, the second stage data is being driven by the COLOR value in which the primitive is to rendered. This COLOR value is received from the system bus data port.
5) Pass (write) the COLOR data out to the VRAM port.

To execute these DRAM/VRAM operations, there are control signals that must be provided externally. The definition of these control signals are as follows:

VSEL	select the RAM port that is to be read and subsequently clocked into the barrel shifter.
BB1_CK	clock the first stage barrel shifter from the selected RAM port (DRAM or VRAM).
BB2_CK	clock the data from the first stage barrel shifter into the second.
VALUOE/	tri-state control for the VRAM port.
DALUOE/	tri-state control for the DRAM port.
ALUCK	clock used to pass data from the second stage barrel shifter (and the RAM port if a logical operation is performed) into the ALU register.
LDCK	clock used to load the shift code for the barrel shifter and the function code for the ALU's.
LD0_SEL	select line used to indicate if the lower two bits or upper two bits of the shift code and function code are being clocked in.
SHIFT(0..1)	shift code input lines for receiving 4-bit shift code.
FUNCTION(0..1)	function code input lines for receiving 4-bit function code.
DRAW	input line for timing pulse to set the COLOR value in the barrel shifter after the WRITE ENABLE value has

	been received externally from the first stage barrel shifter.
SDQ(0..3)	system bus I/O –used to receive the COLOR value when system bus read/writes are not being performed.

Shift Code Generation and Loading

The four bit shift code which is required for a 16 bit barrel shift operation, must be loaded two bits at a time into the Pixel Processor using LDCK and LD0_SEL. First the lower two bits of the shift code are loaded. These bits may be clocked any time before or after the data is clocked into the first stage of the barrel shifter with BB1_CK. However, the shift code must be valid and meet the same set-up time to BB1_CK as the the data at the selected RAM port. This is because these lower two bits are used directly from the input lines as the shift code for the first stage of the barrel shifter.

The second stage of the barrel shifter requires both the lower and upper two bits of the 4-bit shift code to calculate it's two-bit shift code value. Unlike the first stage of the barrel shifter, the second stage uses a registered shift code. Therefore, the upper two bits of the shift code must be clocked by LDCK a period of "T" before BB2_CK so that the following timing constraint is met:

T > t(co) MACH230 + t(su) MACH230 for internal feedback signals.

The shift code that must be provided to the Pixel Processor for BLT operations is calculated as follows:

shift code = destination RAM address(0..3) – source RAM address (0..3)

The shift code that must be provided to the Pixel Processor for system read operations is calculated as follows:

shift code(2..3) = 4 – system address of frame buffer
shift code(0..1) = 0.

The system read shift code calculation ensures the 4 following possible groups of pixels to be read:

Shift Code	Pixel Group
0000	0 1 2 3
0100	C D E F
1000	8 9 A B
1100	4 5 6 7

Because the Pixel Processor is oriented across a single plane of multiple pixels, each Pixel Processor will provide a single bit for each of 4 pixels selected from one of the four pixel groups defined above. Therefore, 8 Pixel Processors will provide 32 bits of information, or one byte for each of four pixels.

Function Code Loading

Function code loading is simpler than shift code loading because all four function code bits are registered as they are clocked in by LDCK. They too are loaded two bits at a time. These bits are also specified by LD0_SEL. The only timing constraint is that the four bits are loaded a period "T" before ALUCK where

T > t(co) MACH230 + t(su) MACH230 for internal feedback signals.

This is necessary because the ALU must see the function code and its operands before it clocks the result.

System Read/Write Interface

The Pixel Processor provides a separate system interface to allow read and write operations to the frame buffers without the need to be connected directly to either of the RAM ports.

Due to the large number of pins required for the Pixel Processor, the COLOR value which is used by some pixel operations, must also share the system data bus. This bus sharing is acceptable since system accesses are performed independently of pixel operations.

Besides the four system bus bits for passing data, there are also 2 control lines. There names and definitions are as follows:

Signal Name	Signal Definition
SENBL	system access enable is used to force the data values seen on the system bus (SDQ(0..3)) into the primary pixel data path within the Pixel Processor. More specifically, the four data bits are used to set or clear each of the four 4:4 barrel shifter in the second stage barrel shifter. SENBL is also an enable signal for driving the system bus out from the Pixel Processor.
SOE/	system bus output enable is used for reading the data out of the Pixel Processor for a system read access of the frame buffers.

Write Enable Interface

The final bus is the write enable interface. Like the system data bus, this interface provides an independent way to access information read from the frame buffers. In this case, the data read from the source RAM is used to form a write mask to the destination RAM.

Because there are other possible operations that must be performed on this write enable data before the final destination write mask, it is necessary to pass this information out of the Pixel Processor at the earliest possible time. Therefore, the data is passed out from the output of the first stage barrel shifter.

Ideally, the completed shifted data should be passed out. However, in COLOR write cases, the case which uses the write enable data, the second stage barrel shifter is forced to receive the COLOR value rather than to finish the shift operation. Because the shift operation is incomplete, the write enable mask passed out of the Pixel Processor must undergo a second stage shift operation.

In order to anti-alias, a number of bit planes (rather than just one) are required to form the graphic primitive template. Anti-aliasing is performed as a write-only pixel operation. This is acceptable for many application in which the background is fairly uniform. The graphic primitive template is concatenated to the COLOR data which is now less than 8 bits. Together, however, they form an 8 bit color value.

To understand the way anti-aliasing is now supported, three facts must be understood. Then, an example will be given:

1) Each Pixel Processor operates on only one plane.
2) Therefore, only pixel operations that can be done on a "per bit" level can be performed. ie. no pixel adding or comparing (which requires carry operations between bits).
3) Anti-aliasing requires that at least one of the modulation bits be non-zero in order to write a pixel.

Therefore, because any one Pixel Processor can only operate on one of the modulation bits, there must be another way to generate a write operation. This other way is shown below by example:

1) When storing the template, use 3 bits (for example) for modulation.
2) Use a fourth bit as the write bit of the template.

Bit	Template	Pixel Color	Result
0	modulation bit 0	X	modulation bit 0
1	modulation bit 1	X	modulation bit 1
2	modulation bit 2	X	modulation bit 2
3	template write bit	COLOR bit 0	COLOR bit 0
4	X	COLOR bit 1	COLOR bit 1
5	X	COLOR bit 2	COLOR bit 2
6	X	COLOR bit 3	COLOR bit 3
7	X	COLOR bit 4	COLOR bit 4

X = don't care.

Example 1.

Bit	Template	Pixel Color	Result
0	0	X	0
1	1	X	1
2	0	X	0
3	1	0	0
4		1	1
5		1	1
6		1	1
7		1	1

Example 2.

Bit	Template	Pixel Color	Result
0	0	X	No Write
1	0	X	
2	0	X	
3	0	0	
4		1	
5		1	
6		1	
7		1	

Write Enable Generation Interface–MIXING CONTROLLER MACH PAL

The MIXING CONTROLLER is designed together with the Pixel Processor to take advantage of the plane-wise division of the pixel operations. The function of MIXING CONTROLLER is to generate WE's to the VRAM's and DRAM's. The WE's are generated by AND'ing WE's from three separate sources. The MIXING CONTROLLER does not distinguish between DRAM and VRAM WE's. It generates only one set for both. After this, the WE's are enabled at either the VRAM or DRAM.

The three WE sources that the MIXING CONTROLLER uses are PP WE (from Pixel Processor), PP fifo WE (used to hold stipple patterns), and clip mask/system bus WE. These three sources are AND'ed together to make the final WE to the RAM's. The three sources may be independently enabled or disabled as the design permits.

CONCLUSION

The Pixel Processor design, as well as the other design conversions that were made to the original PAL-based design, were all successful and in time without surprises. This happy outcome was a combination of the tools, the MACH architecture, and luck.

The MACH family parts provide the flexibility, density, and speed for chip functions normally relegated to smaller gate arrays. The uniformity in the MACH architecture provides the basis for fast design and prototyping as well.

ACKNOWLEDGMENTS

Special thanks to Atsushi Hakamata of SONY Corporation (Osaki Technology Center-Tokyo) and Mark McClain of Advanced Micro Devices (San Diego Office) for their invaluable assistance through the product development.

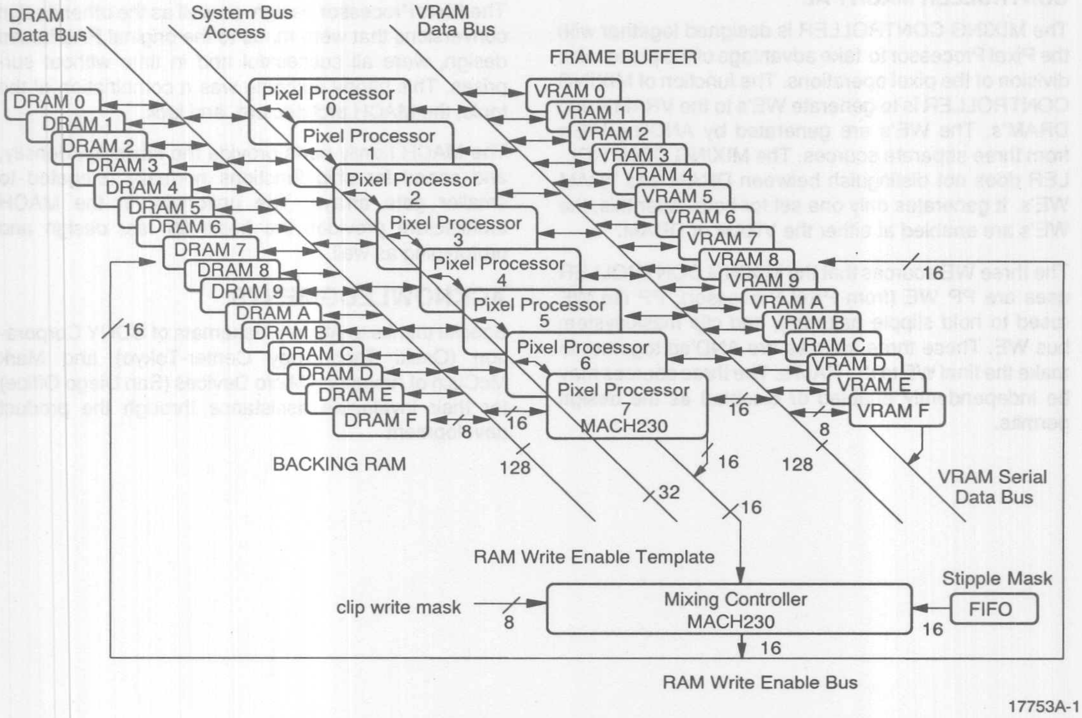


Figure 9-1. Pixel Processor System Diagram

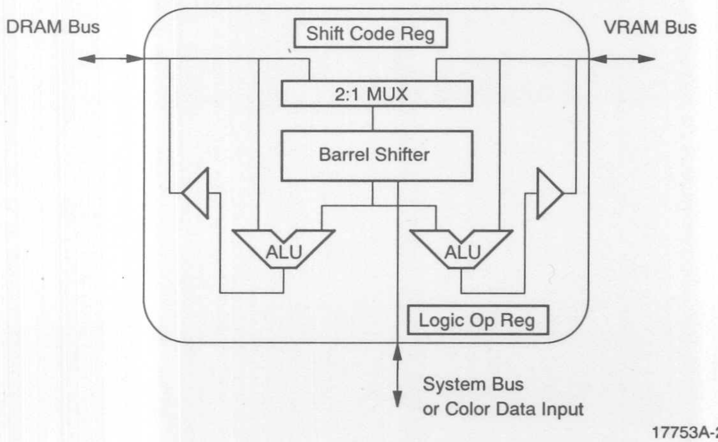
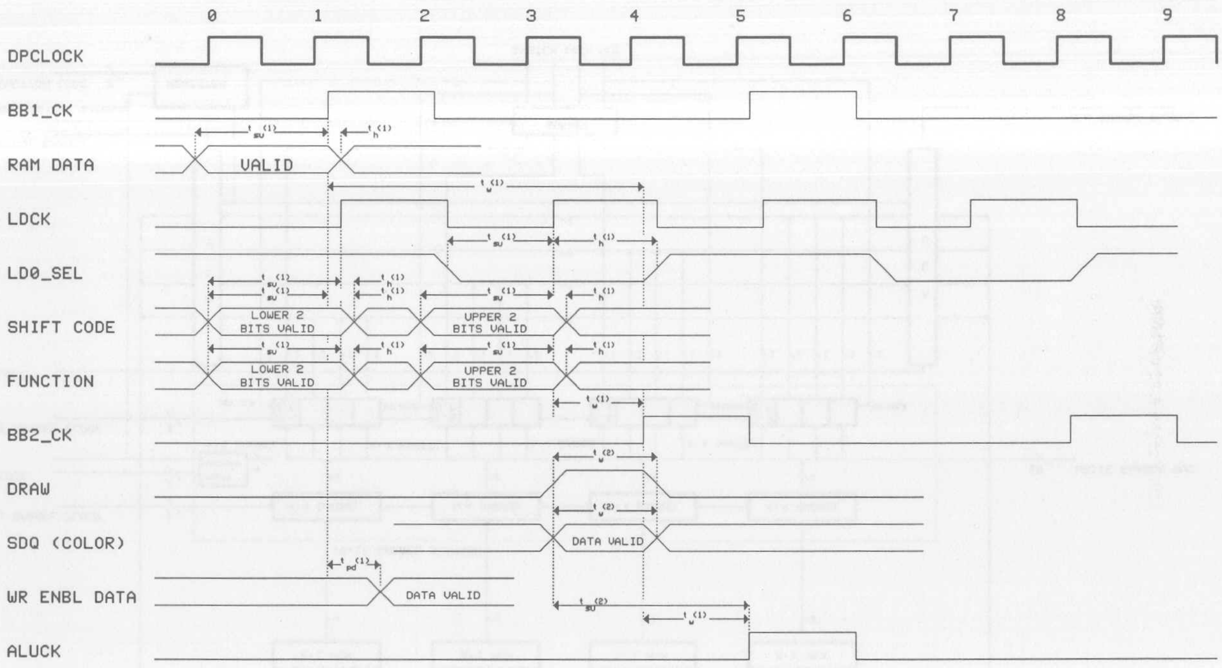


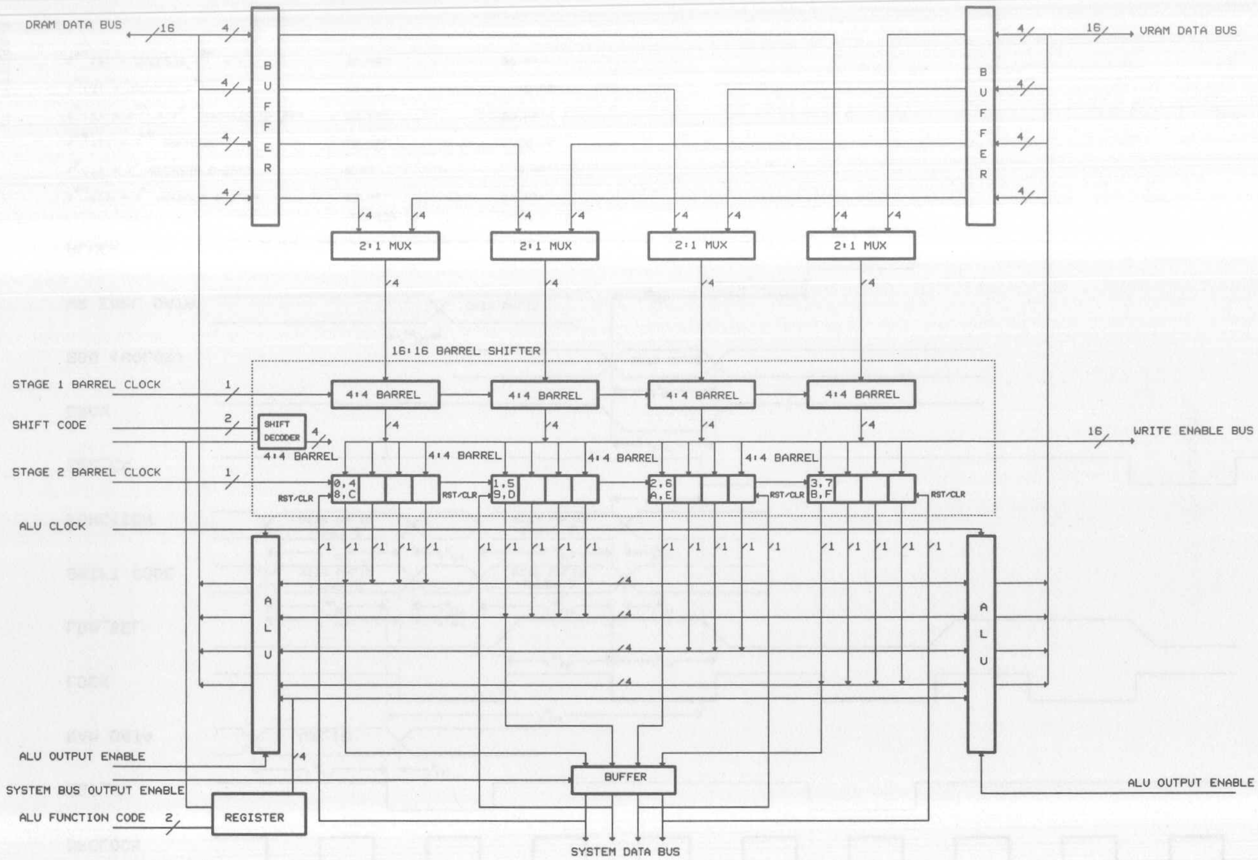
Figure 9-2. Pixel Processor Block Diagram



	MACH230-20	MACH230-15
$t_{su}(1) = t_s$ MACH230 D-Type	13 ns.	10 ns.
$t_h(1) = t_h$ MACH230 D-Type	0 ns.	0 ns.
$t_{pd}(1) = t_{co}$ MACH230	12 ns.	10 ns.
$t_w(1) = t_{co} + t_s$ MACH230 D-Type	25 ns.	20 ns.
$t_w(2)$ = MACH230 t_{arw}	20 ns.	15 ns.
$t_{su}(2)$ = MACH230 $t_{ap} + t_s$	38 ns.	30 ns.

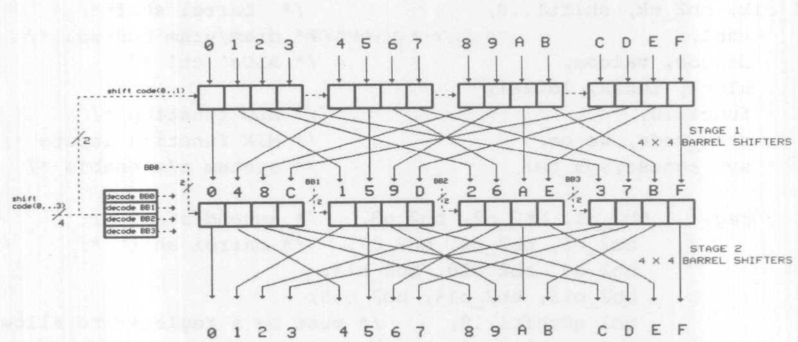
Figure 9-3. Pixel Processor Timing Specification

17753A-3



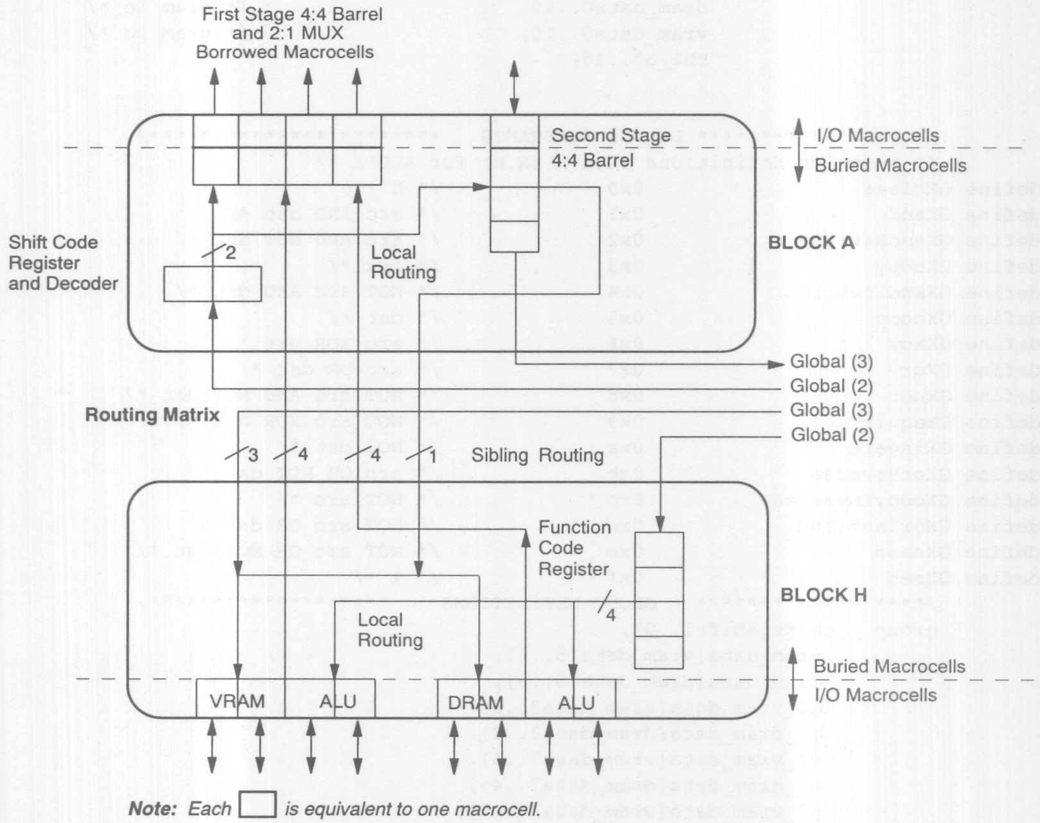
17753A-4

Figure 9-4. Pixel Processor Detail



17753A-5

Figure 9-5. Pixel Processor 16x16 Barrel Shifter Detail



17753A-6

Figure 9-6. Fitting 4 Bits of Data Functions into 2 MACH230 Blocks

SOURCE FILE

```
pp_brl   (in bb1_clk, bb2_ck, shift1..0,              /*  barrel shift*/
                  vsel,                               /* dram/vram control */
                  daluoe, valuoe,                     /* ALUs' ctl */
                  aluck, ldclk, ld0sel,
                  func1..0,                           /* ALU function */
                  draw_mode, we_oe,                   /* MIX function inputs */
                  sys_access,sys_oe;                  /* system r/w enable */

                  reg     bb2_o1, bb2_o2, bb2_o3,     /* second stage */
                           bb2_o5, bb2_o6, bb2_o7,     /* barrel shift */
                           bb2_o9, bb2_o10, bb2_o11,
                           bb2_o13, bb2_o14, bb2_o15;
                           bb2_g0shft1..0,        /* must be a register to allow */
                           bb2_g1shft1..0,        /* for pre,clr from COLOR input */
                           bb2_g2shft1..0,
                           bb2_g3shft1..0,
                           g0_func3..0, g1_func3..0, g2_func3..0, g3_func3..0;
                  io reg  bb2_o0, bb2_o4, bb2_o8, bb2_o12, /* system bus io
                                                               and barrel stage 2*/
                           dram_data0..15,                     /* dram io */
                           vram_data0..15,                     /* vram io */
                           bb1_o0..15)

{

         /****************** DEFINE CONSTANTS   **********************/
         /* function definitions from X (X.h) for ALU's */
#define GXclear               0x0           /* 0 */
#define GXand                 0x1           /* src AND dst */
#define GXandReverse          0x2           /* src AND NOT dst */
#define GXcopy                0x3           /* src */
#define GXandInverted         0x4           /* NOT src AND dst */
#define GXnoop                0x5           /* dst */
#define GXxor                 0x6           /* src XOR dst */
#define GXor                  0x7           /* src OR dst */
#define GXnor                 0x8           /* NOT src AND NOT dst */
#define GXequiv               0x9           /* NOT src XOR dst */
#define GXinvert              0xa           /* NOT dst */
#define GXorReverse           0xb           /* src OR NOT dst */
#define GXcopyInverted        0xc           /* NOT src */
#define GXorInverted          0xd           /* NOT src OR dst */
#define GXnand                0xe           /* NOT src OR NOT dst */
#define GXset                 0xf           /* 1 */
         /******************   GROUP DEFINITIONS      ******************/
         group   shift[shift1..0],
                  vram_data[vram_data15..0],
                  dram_data[dram_data15..0],
                  g0_vram_data[vram_data3..0],
                  g0_dram_data[dram_data3..0],
                  g1_vram_data[vram_data7..4],
                  g1_dram_data[dram_data7..4],
                  g2_vram_data[vram_data11..8],
                  g2_dram_data[dram_data11..8],
                  g3_vram_data[vram_data15..12],
                  g3_dram_data[dram_data15..12],
                  func[func1..0],
```

```
g0_func[g0_func3..0],
g1_func[g1_func3..0],
g2_func[g2_func3..0],
g3_func[g3_func3..0],         /* stage 1 barrel groups */

bb1_d[dram_data15..0],
bb1_v[vram_data15..0],
bb1_o[bb1_o15..0],
                                        /* bb1 inputs */
bb1_g0d[dram_data03..0],
bb1_g1d[dram_data07..4],
bb1_g2d[dram_data11..8],
bb1_g3d[dram_data15..12],

bb1_g0v[vram_data03..0],
bb1_g1v[vram_data07..4],
bb1_g2v[vram_data11..8],
bb1_g3v[vram_data15..12],

bb1_g0o[bb1_o03..0],                /* bb1 outputs */
bb1_g1o[bb1_o07..4],
bb1_g2o[bb1_o11..8],
bb1_g3o[bb1_o15..12],
                                        /* stage 2 barrel groups */
                                        /* bb2 inputs */
bb2_g0d[bb1_o12, bb1_o8, bb1_o4, bb1_o0],
bb2_g1d[bb1_o13, bb1_o9, bb1_o5, bb1_o1],
bb2_g2d[bb1_o14, bb1_o10, bb1_o6, bb1_o2],
bb2_g3d[bb1_o15, bb1_o11, bb1_o7, bb1_o3],
                                    /* bb2 outputs */
bb2_o[bb2_o15, bb2_o14, bb2_o13, bb2_o12,
      bb2_o11, bb2_o10, bb2_o9, bb2_o8,
      bb2_o7, bb2_o6, bb2_o5, bb2_o4,
      bb2_o3, bb2_o2, bb2_o1, bb2_o0],
bb2_g0o[bb2_o3, bb2_o2, bb2_o1, bb2_o0],
bb2_g1o[bb2_o7, bb2_o6, bb2_o5, bb2_o4],
bb2_g2o[bb2_o11, bb2_o10, bb2_o9, bb2_o8],
bb2_g3o[bb2_o15, bb2_o14, bb2_o13, bb2_o12],

/* bb2 outputs ordered for inputs to dram/vram ALU's */
alu_d[bb2_o15, bb2_o11, bb2_o7, bb2_o3,
      bb2_o14, bb2_o10, bb2_o6, bb2_o2,
      bb2_o13, bb2_o9, bb2_o5, bb2_o1,
      bb2_o12, bb2_o8, bb2_o4, bb2_o0],
alu_g0d[bb2_o12, bb2_o8, bb2_o4, bb2_o0],
alu_g1d[bb2_o13, bb2_o9, bb2_o5, bb2_o1],
alu_g2d[bb2_o14, bb2_o10, bb2_o6, bb2_o2],
alu_g3d[bb2_o15, bb2_o11, bb2_o7, bb2_o3],
                        /* sysdata is first bit of each stage 2
                        4:4 barrel shifter */
sysdata[bb2_o12, bb2_o8, bb2_o4,  bb2_o0],
                                        /* bb2 shift codes */
bb2_g0shft[bb2_g0shft1..0],
bb2_g1shft[bb2_g1shft1..0],
bb2_g2shft[bb2_g2shft1..0],
```

```
        bb2_g3shft[bb2_g3shft1..0];

/***************  GLOBAL CLOCK,  RESET, AND OE's ************/
/* barrel shifter definitions */
bb1_o.ck = bb1_clk;
bb1_o.oe = !we_oe;
bb2_o.ck = bb2_clk;

/* These rst's and clr's will probably not be used anymore since it
resets everything in the block (including bb1). Use a special decode
for bb2_o[].i as inputs to the data that's latched and passed
to the ALU's for system writes. only one bit per 4:4 barrel */

bb2_g0shft.ck = ldclk;
bb2_g1shft.ck = ldclk;
bb2_g2shft.ck = ldclk;
bb2_g3shft.ck = ldclk;

/* pixel ALU definitions */
vram_data.oe = !valuoe;
dram_data.oe = !daluoe;
vram_data.ck = aluck;
dram_data.ck = aluck;
g0_func.ck = ldclk;
g1_func.ck = ldclk;
g2_func.ck = ldclk;
g3_func.ck = ldclk;

/**********************  SYSTEM BUS   ******************/
/* The system bus fram buffer reads are now done through the first bit
of each 4:4 group in the second stage of the barrel shifter. The shift
code becomes the address but will not account for de-interleaving.
This must be done externally when the sysadr is driven onto the shift
code inputs. */

/* system bus read */
sysdata[ ].oe = sys_oe & sys_access;
/* system bus write and color write */
bb2_g0o[].apre = bb2_o0.i & draw_mode & !sys_oe;
bb2_g0o[].aclr = !bb2_o0.i & draw_mode & !sys_oe;
bb2_g1o[].apre = bb2_o4.i & draw_mode & !sys_oe;
bb2_g1o[].aclr = !bb2_o4.i & draw_mode & !sys_oe;
bb2_g2o[].apre = bb2_o8.i & draw_mode & !sys_oe;
bb2_g2o[].aclr = !bb2_o8.i & draw_mode & !sys_oe;
bb2_g3o[].apre = bb2_o12.i & draw_mode & !sys_oe;
bb2_g3o[].aclr = !bb2_o12.i & draw_mode & !sys_oe;

/****************** SYSTEM FUNCTION  AND CONTROL  **********/
/* ALU function load */

if (ld0sel) {
        g3_func[0]=func[0];
        g3_func[1]=func[1];
        g3_func[2]=g3_func[2];
        g3_func[3]=g3_func[3];
```

```
}
else {
        g3_func[0]=g3_func[0];
        g3_func[1]=g3_func[1];
        g3_func[2]=func[0];
        g3_func[3]=func[1];
}
if (ld0sel) {
        g2_func[0]=func[0];
        g2_func[1]=func[1];
        g2_func[2]=g2_func[2];
        g2_func[3]=g2_func[3];
}
else {
        g2_func[0]=g2_func[0];
        g2_func[1]=g2_func[1];
        g2_func[2]=func[0];
        g2_func[3]=func[1];
}
if (ld0sel) {
        g1_func[0]=func[0];
        g1_func[1]=func[1];
        g1_func[2]=g1_func[2];
        g1_func[3]=g1_func[3];
}
else {
        g1_func[0]=g1_func[0];
        g1_func[1]=g1_func[1];
        g1_func[2]=func[0];
        g1_func[3]=func[1];
}
if (ld0sel) {
        g0_func[0]=func[0];
        g0_func[1]=func[1];
        g0_func[2]=g0_func[2];
        g0_func[3]=g0_func[3];
}
else {
        g0_func[0]=g0_func[0];
        g0_func[1]=g0_func[1];
        g0_func[2]=func[0];
        g0_func[3]=func[1];
}
/*************  16:16 BARREL SHIFTER *****************/
/* Stage 1 equations */
if (vsel) {

    bb1_g0o = (bb1_g0v[].i << shift) | (bb1_g0v[].i >> (4 - shift));
    bb1_g1o = (bb1_g1v[].i << shift) | (bb1_g1v[].i >> (4 - shift));
    bb1_g2o = (bb1_g2v[].i << shift) | (bb1_g2v[].i >> (4 - shift));
    bb1_g3o = (bb1_g3v[].i << shift) | (bb1_g3v[].i >> (4 - shift));
}
else {
    bb1_g0o = (bb1_g0d[].i << shift) | (bb1_g0d[].i >> (4 -shift));
    bb1_g1o = (bb1_g1d[].i << shift) | (bb1_g1d[].i >> (4 -shift));
    bb1_g2o = (bb1_g2d[].i << shift) | (bb1_g2d[].i >> (4 -shift));
```

```
    bb1_g3o = (bb1_g3d[].i << shift) | (bb1_g3d[].i >> (4 -shift));
}

/* stage 2 equations */
/* calculate stage 2 shift codes */

if (ld0sel) { /* load first operand (low 2 bits shift code) for
                  2nd stage shift code calculation */
         bb2_g0shft = shift[];
         bb2_g1shft = shift[];
         bb2_g2shft = shift[];
         bb2_g3shft = shift[];
}
else {    /* calculate code for stage 2*/
         bb2_g3shft[]=shift[];
         if (bb2_g0shft>0) bb2_g0shft[]=shift[]+1;
         else          bb2_g0shft[]=shift[];
         if (bb2_g1shft>1) bb2_g1shft[]=shift[]+1;
         else          bb2_g1shft[]=shift;
         if (bb2_g2shft==3) bb2_g2shft[]=shift[]+1;
         else          bb2_g2shft[]=shift[];
}
/* stage 2 shift equations */
bb2_g0o = (bb2_g0d << bb2_g0shft) | (bb2_g0d >> (4 - bb2_g0shft));
bb2_g1o = (bb2_g1d << bb2_g1shft) | (bb2_g1d >> (4 - bb2_g1shft));
bb2_g2o = (bb2_g2d << bb2_g2shft) | (bb2_g2d >> (4 - bb2_g2shft));
bb2_g3o = (bb2_g3d << bb2_g3shft) | (bb2_g3d >> (4 - bb2_g3shft));
/******************   ALU's ***************/
/* DRAM ALU */
         switch(g0_func)
         {
                  case GXclear:
                           g0_dram_data = 0;
                           break;
                  case GXnor:
                           g0_dram_data = alu_g0d.fb !| g0_dram_data.i;
                           break;
                  case GXandInverted:
                           g0_dram_data = ~alu_g0d.fb & g0_dram_data.i;
                           break;
                  case GXcopyInverted:
                           g0_dram_data = ~alu_g0d.fb;
                           break;
                  case GXandReverse:
                           g0_dram_data = alu_g0d.fb & ~g0_dram_data.i;
                           break;
                  case GXinvert:
                           g0_dram_data = ~g0_dram_data.i;
                           break;
                  case GXxor:
                           g0_dram_data = alu_g0d.fb ^ g0_dram_data.i;
                           break;
                  case GXnand:
                           g0_dram_data = alu_g0d.fb !& g0_dram_data.i;
                           break;
                  case GXand:
```

```
                        g0_dram_data = alu_g0d.fb & g0_dram_data.i;
                        break;
                case GXequiv:
                        g0_dram_data = alu_g0d.fb !^ g0_dram_data.i;
                        break;
                case GXnoop:
                        g0_dram_data = g0_dram_data.i;
                        break;
                case GXorInverted:
                        g0_dram_data = ~alu_g0d.fb | g0_dram_data.i;
                        break;
                case GXcopy:
                        g0_dram_data = alu_g0d.fb;
                        break;
                case GXorReverse:
                        g0_dram_data = alu_g0d.fb | ~g0_dram_data.i;
                        break;
                case GXor:
                        g0_dram_data = alu_g0d.fb | g0_dram_data.i;
                        break;
                case GXset:
                        g0_dram_data = HI;
                        break;
        }   /* end dram ALU switch */
/* VRAM ALU */
        switch(g0_func)
        {
                case GXclear:
                        g0_vram_data = 0;
                        break;
                case GXnor:
                        g0_vram_data = alu_g0d.fb !| g0_vram_data.i;
                        break;
                case GXandInverted:
                        g0_vram_data = ~alu_g0d.fb & g0_vram_data.i;
                        break;
                case GXcopyInverted:
                        g0_vram_data = ~alu_g0d.fb;
                        break;
                case GXandReverse:
                        g0_vram_data = alu_g0d.fb & ~g0_vram_data.i;
                        break;
                case GXinvert:
                        g0_vram_data = ~g0_vram_data.i;
                        break;
                case GXxor:
                        g0_vram_data = alu_g0d.fb ^ g0_vram_data.i;
                        break;
                case GXnand:
                        g0_vram_data = alu_g0d.fb !& g0_vram_data.i;
                        break;
                case GXand:
                        g0_vram_data = alu_g0d.fb & g0_vram_data.i;
                        break;
                case GXequiv:
                        g0_vram_data = alu_g0d.fb !^ g0_vram_data.i;
```

```
                        break;
                case GXnoop:
                        g0_vram_data = g0_vram_data.i;
                        break;
                case GXorInverted:
                        g0_vram_data = ~alu_g0d.fb | g0_vram_data.i;
                        break;
                case GXcopy:
                        g0_vram_data = alu_g0d.fb;
                        break;
                case GXorReverse:
                        g0_vram_data = alu_g0d.fb | ~g0_vram_data.i;
                        break;
                case GXor:
                        g0_vram_data = alu_g0d.fb | g0_vram_data.i;
                        break;
                case GXset:
                        g0_vram_data = HI;
                        break;
            }   /* end vram ALU switch */
/* End GROUP 0 ALU equations */
/* GROUP 1 ALU equations here */
/* GROUP 2 ALU equations here */
/* GROUP 3 ALU equations here */
```

DESIGN FILE

```
;PALASM Design Description
;------------------------------- Declaration Segment ------
TITLE        PP
PATTERN      PP
REVISION     0.1
AUTHOR       Ray_Broemmelsiek
COMPANY      SONY
DATE         2/5/92
CHIP   _PP  Mach230
;------------------------------- PIN Declarations ---------
PIN 7 pbb1_o15 REG ; IO
PIN 5 pbb1_o14 REG ; IO
PIN 6 pbb1_o13 REG ; IO
PIN 4 pbb1_o12 REG ; IO
PIN 15 pbb1_o11 REG ; IO
PIN 17 pbb1_o10 REG ; IO
PIN 16 pbb1_o9 REG ; IO
PIN 18 pbb1_o8 REG ; IO
PIN 27 pbb1_o7 REG ; IO
PIN 26 pbb1_o6 REG ; IO
PIN 25 pbb1_o5 REG ; IO
PIN 24 pbb1_o4 REG ; IO
PIN 36 pbb1_o3 REG ; IO
PIN 37 pbb1_o2 REG ; IO
PIN 38 pbb1_o1 REG ; IO
PIN 39 pbb1_o0 REG ; IO
NODE 10 bb1_o15 PAIR pbb1_o15 REGISTERED ;
NODE 6 bb1_o14 PAIR pbb1_o14 REGISTERED ;
NODE 8 bb1_o13 PAIR pbb1_o13 REGISTERED ;
NODE 4 bb1_o12 PAIR pbb1_o12 REGISTERED ;
NODE 26 bb1_o11 PAIR pbb1_o11 REGISTERED ;
NODE 22 bb1_o10 PAIR pbb1_o10 REGISTERED ;
NODE 24 bb1_o9 PAIR pbb1_o9 REGISTERED ;
NODE 20 bb1_o8 PAIR pbb1_o8 REGISTERED ;
NODE 40 bb1_o7 PAIR pbb1_o7 REGISTERED ;
NODE 38 bb1_o6 PAIR pbb1_o6 REGISTERED ;
NODE 36 bb1_o5 PAIR pbb1_o5 REGISTERED ;
NODE 34 bb1_o4 PAIR pbb1_o4 REGISTERED ;
NODE 58 bb1_o3 PAIR pbb1_o3 REGISTERED ;
NODE 56 bb1_o2 PAIR pbb1_o2 REGISTERED ;
NODE 54 bb1_o1 PAIR pbb1_o1 REGISTERED ;
NODE 52 bb1_o0 PAIR pbb1_o0 REGISTERED ;
PIN 76 pvram_data15 REG ; IO
PIN 80 pvram_data14 REG ; IO
PIN 81 pvram_data13 REG ; IO
PIN 82 pvram_data12 REG ; IO
PIN 73 pvram_data11 REG ; IO
PIN 69 pvram_data10 REG ; IO
PIN 67 pvram_data9 REG ; IO
PIN 66 pvram_data8 REG ; IO
PIN 60 pvram_data7 REG ; IO
PIN 61 pvram_data6 REG ; IO
PIN 54 pvram_data5 REG ; IO
PIN 56 pvram_data4 REG ; IO
PIN 52 pvram_data3 REG ; IO
```

```
PIN 51 pvram_data2 REG ; IO
PIN 50 pvram_data1 REG ; IO
PIN 48 pvram_data0 REG ; IO
NODE 126 vram_data15 PAIR pvram_data15 REGISTERED ;
NODE 118 vram_data14 PAIR pvram_data14 REGISTERED ;
NODE 116 vram_data13 PAIR pvram_data13 REGISTERED ;
NODE 114 vram_data12 PAIR pvram_data12 REGISTERED ;
NODE 112 vram_data11 PAIR pvram_data11 REGISTERED ;
NODE 104 vram_data10 PAIR pvram_data10 REGISTERED ;
NODE 100 vram_data9 PAIR pvram_data9 REGISTERED ;
NODE 98 vram_data8 PAIR pvram_data8 REGISTERED ;
NODE 84 vram_data7 PAIR pvram_data7 REGISTERED ;
NODE 82 vram_data6 PAIR pvram_data6 REGISTERED ;
NODE 96 vram_data5 PAIR pvram_data5 REGISTERED ;
NODE 92 vram_data4 PAIR pvram_data4 REGISTERED ;
NODE 80 vram_data3 PAIR pvram_data3 REGISTERED ;
NODE 78 vram_data2 PAIR pvram_data2 REGISTERED ;
NODE 76 vram_data1 PAIR pvram_data1 REGISTERED ;
NODE 72 vram_data0 PAIR pvram_data0 REGISTERED ;
PIN 77 pdram_data15 REG ; IO
PIN 78 pdram_data14 REG ; IO
PIN 75 pdram_data13 REG ; IO
PIN 79 pdram_data12 REG ; IO
PIN 71 pdram_data11 REG ; IO
PIN 70 pdram_data10 REG ; IO
PIN 72 pdram_data9 REG ; IO
PIN 68 pdram_data8 REG ; IO
PIN 55 pdram_data7 REG ; IO
PIN 58 pdram_data6 REG ; IO
PIN 57 pdram_data5 REG ; IO
PIN 59 pdram_data4 REG ; IO
PIN 46 pdram_data3 REG ; IO
PIN 49 pdram_data2 REG ; IO
PIN 47 pdram_data1 REG ; IO
PIN 45 pdram_data0 REG ; IO
NODE 124 dram_data15 PAIR pdram_data15 REGISTERED ;
NODE 122 dram_data14 PAIR pdram_data14 REGISTERED ;
NODE 128 dram_data13 PAIR pdram_data13 REGISTERED ;
NODE 120 dram_data12 PAIR pdram_data12 REGISTERED ;
NODE 108 dram_data11 PAIR pdram_data11 REGISTERED ;
NODE 106 dram_data10 PAIR pdram_data10 REGISTERED ;
NODE 110 dram_data9 PAIR pdram_data9 REGISTERED ;
NODE 102 dram_data8 PAIR pdram_data8 REGISTERED ;
NODE 94 dram_data7 PAIR pdram_data7 REGISTERED ;
NODE 88 dram_data6 PAIR pdram_data6 REGISTERED ;
NODE 90 dram_data5 PAIR pdram_data5 REGISTERED ;
NODE 86 dram_data4 PAIR pdram_data4 REGISTERED ;
NODE 68 dram_data3 PAIR pdram_data3 REGISTERED ;
NODE 74 dram_data2 PAIR pdram_data2 REGISTERED ;
NODE 70 dram_data1 PAIR pdram_data1 REGISTERED ;
NODE 66 dram_data0 PAIR pdram_data0 REGISTERED ;
PIN 40 pbb2_o12 REG ; IO
PIN 29 pbb2_o8 REG ; IO
PIN 19 pbb2_o4 REG ; IO
PIN 3 pbb2_o0 REG ; IO
NODE 50 bb2_o12 PAIR pbb2_o12 REGISTERED ; IO
```

```
NODE 44 bb2_o8 PAIR pbb2_o8 REGISTERED ; IO
NODE 18 bb2_o4 PAIR pbb2_o4 REGISTERED ; IO
NODE 2 bb2_o0 PAIR pbb2_o0 REGISTERED ; IO
NODE 117 g3_func0 REGISTERED ; IO
NODE 119 g3_func1 REGISTERED ; IO
NODE 121 g3_func2 REGISTERED ; IO
NODE 115 g3_func3 REGISTERED ; IO
NODE 99 g2_func0 REGISTERED ; IO
NODE 101 g2_func1 REGISTERED ; IO
NODE 103 g2_func2 REGISTERED ; IO
NODE 105 g2_func3 REGISTERED ; IO
NODE 83 g1_func0 REGISTERED ; IO
NODE 85 g1_func1 REGISTERED ; IO
NODE 87 g1_func2 REGISTERED ; IO
NODE 89 g1_func3 REGISTERED ; IO
NODE 67 g0_func0 REGISTERED ; IO
NODE 69 g0_func1 REGISTERED ; IO
NODE 71 g0_func2 REGISTERED ; IO
NODE 73 g0_func3 REGISTERED ; IO
NODE 59 bb2_g3shft0 REGISTERED ; IO
NODE 57 bb2_g3shft1 REGISTERED ; IO
NODE 47 bb2_g2shft0 REGISTERED ; IO
NODE 43 bb2_g2shft1 REGISTERED ; IO
NODE 27 bb2_g1shft0 REGISTERED ; IO
NODE 29 bb2_g1shft1 REGISTERED ; IO
NODE 13 bb2_g0shft0 REGISTERED ; IO
NODE 11 bb2_g0shft1 REGISTERED ; IO
NODE 64 bb2_o15 REGISTERED ; IO
NODE 62 bb2_o14 REGISTERED ; IO
NODE 60 bb2_o13 REGISTERED ; IO
NODE 46 bb2_o11 REGISTERED ; IO
NODE 42 bb2_o10 REGISTERED ; IO
NODE 45 bb2_o9 REGISTERED ; IO
NODE 28 bb2_o7 REGISTERED ; IO
NODE 25 bb2_o6 REGISTERED ; IO
NODE 30 bb2_o5 REGISTERED ; IO
NODE 15 bb2_o3 REGISTERED ; IO
NODE 14 bb2_o2 REGISTERED ; IO
NODE 12 bb2_o1 REGISTERED ; IO
PIN 30 sys_oe ; INPUT
PIN 31 sys_access ; INPUT
PIN 14 we_oe ; INPUT
PIN 13 draw_mode ; INPUT
PIN 35 func0 ; INPUT
PIN 34 func1 ; INPUT
PIN 41 ld0sel ; INPUT
PIN 65 ldclk ; INPUT
PIN 62 aluck ; INPUT
PIN 28 valuoe ; INPUT
PIN 83 daluoe ; INPUT
PIN 12 vsel ; INPUT
PIN 9 shift0 ; INPUT
PIN 8 shift1 ; INPUT
PIN 23 bb_clk ; INPUT
PIN 20 bb2_clk ; INPUT
GROUP    MACH_SEG_A      bb1_o0  bb1_o1   bb1_o2 bb1_o3
```

```
bb2_o0 bb2_o1  bb2_o2   bb2_o3
bb2_g0shft0    bb2_g0shft1
GROUP   MACH_SEG_B       bb1_o4  bb1_o5   bb1_o6 bb1_o7
bb2_o4 bb2_o5  bb2_o6   bb2_o7
bb2_g1shft0    bb2_g1shft1
GROUP   MACH_SEG_C       bb1_o8  bb1_o9   bb1_o10 bb1_o11
bb2_o8 bb2_o9  bb2_o10   bb2_o11
bb2_g2shft0    bb2_g2shft1
GROUP   MACH_SEG_D       bb1_o12  bb1_o13    bb1_o14 bb1_o15
bb2_o12 bb2_o13  bb2_o14   bb2_o15
bb2_g3shft0    bb2_g3shft1
GROUP   MACH_SEG_H
dram_data3     vram_data3     dram_data2     vram_data2
dram_data1     vram_data1     dram_data0     dram_data0
g0_func0        g0_func1        g0_func2        g0_func3
GROUP   MACH_SEG_G
dram_data7     dram_data6     dram_data5     dram_data4
vram_data7     vram_data6     vram_data5     vram_data4
g1_func0        g1_func1        g1_func2        g1_func3
GROUP   MACH_SEG_F
dram_data8     dram_data10     dram_data9     dram_data11
vram_data8     vram_data10     vram_data9     vram_data11
g2_func0        g2_func1        g2_func2        g2_func3
GROUP   MACH_SEG_E
dram_data12     vram_data12     dram_data13     vram_data13
dram_data14     vram_data14     dram_data15     vram_data15
g3_func0        g3_func1        g3_func2        g3_func3
;------------------------------- Boolean Equation Segment -
EQUATIONS
;*********the bb1_oxx.trst must be removed ****
pbb1_o0.trst = /we_oe ;
pbb1_o1.trst = /we_oe ;
pbb1_o2.trst = /we_oe ;
pbb1_o3.trst = /we_oe ;
pbb1_o4.trst = /we_oe ;
pbb1_o5.trst = /we_oe ;
pbb1_o6.trst = /we_oe ;
pbb1_o7.trst = /we_oe ;
pbb1_o8.trst = /we_oe ;
pbb1_o9.trst = /we_oe ;
pbb1_o10.trst = /we_oe ;
pbb1_o11.trst = /we_oe ;
pbb1_o12.trst = /we_oe ;
pbb1_o13.trst = /we_oe ;
pbb1_o14.trst = /we_oe ;
pbb1_o15.trst = /we_oe ;
;*****************the vram_dataxx.trst equations must be changed or deleted
pvram_data0.trst = /valuoe ;
pvram_data1.trst = /valuoe ;
pvram_data2.trst = /valuoe ;
pvram_data3.trst = /valuoe ;
pvram_data4.trst = /valuoe ;
pvram_data5.trst = /valuoe ;
pvram_data6.trst = /valuoe ;
pvram_data7.trst = /valuoe ;
pvram_data8.trst = /valuoe ;
```

```
pvram_data9.trst = /valuoe ;
pvram_data10.trst = /valuoe ;
pvram_data11.trst = /valuoe ;
pvram_data12.trst = /valuoe ;
pvram_data13.trst = /valuoe ;
pvram_data14.trst = /valuoe ;
pvram_data15.trst = /valuoe ;
pdram_data0.trst = /daluoe ;
pdram_data1.trst = /daluoe ;
pdram_data2.trst = /daluoe ;
pdram_data3.trst = /daluoe ;
pdram_data4.trst = /daluoe ;
pdram_data5.trst = /daluoe ;
pdram_data6.trst = /daluoe ;
pdram_data7.trst = /daluoe ;
pdram_data8.trst = /daluoe ;
pdram_data9.trst = /daluoe ;
pdram_data10.trst = /daluoe ;
pdram_data11.trst = /daluoe ;
pdram_data12.trst = /daluoe ;
pdram_data13.trst = /daluoe ;
pdram_data14.trst = /daluoe ;
pdram_data15.trst = /daluoe ;
pbb2_o0.trst = sys_oe * sys_access ;
pbb2_o4.trst = sys_oe * sys_access ;
pbb2_o8.trst = sys_oe * sys_access ;
pbb2_o12.trst = sys_oe * sys_access ;
;*****************change the bb1_oxx equations right side to "pvram_data <- vram_data"
; **************************** END added equations to support PALASM IO
;Change bb2_o setf and rstf equations to include pbb2_o 0,4,8,12 on rt. side
;Change bb1o equns to use pdram and pvram on rt. side
; Change dram_data and vram_data eqns to use pdram and pvram _data on rt. side
bb1_o0.clkf = bb_clk ;
bb1_o1.clkf = bb_clk ;
bb1_o2.clkf = bb_clk ;
bb1_o3.clkf = bb_clk ;
bb1_o4.clkf = bb_clk ;
bb1_o5.clkf = bb_clk ;
bb1_o6.clkf = bb_clk ;
bb1_o7.clkf = bb_clk ;
bb1_o8.clkf = bb_clk ;
bb1_o9.clkf = bb_clk ;
bb1_o10.clkf = bb_clk ;
bb1_o11.clkf = bb_clk ;
bb1_o12.clkf = bb_clk ;
bb1_o13.clkf = bb_clk ;
bb1_o14.clkf = bb_clk ;
bb1_o15.clkf = bb_clk ;
bb2_o0.clkf = bb2_clk ;
bb2_o1.clkf = bb2_clk ;
bb2_o2.clkf = bb2_clk ;
bb2_o3.clkf = bb2_clk ;
bb2_o4.clkf = bb2_clk ;
bb2_o5.clkf = bb2_clk ;
bb2_o6.clkf = bb2_clk ;
bb2_o7.clkf = bb2_clk ;
```

```
bb2_o8.clkf = bb2_clk ;
bb2_o9.clkf = bb2_clk ;
bb2_o10.clkf = bb2_clk ;
bb2_o11.clkf = bb2_clk ;
bb2_o12.clkf = bb2_clk ;
bb2_o13.clkf = bb2_clk ;
bb2_o14.clkf = bb2_clk ;
bb2_o15.clkf = bb2_clk ;
bb2_g0shft0.clkf = ldclk ;
bb2_g0shft1.clkf = ldclk ;
bb2_g1shft0.clkf = ldclk ;
bb2_g1shft1.clkf = ldclk ;
bb2_g2shft0.clkf = ldclk ;
bb2_g2shft1.clkf = ldclk ;
bb2_g3shft0.clkf = ldclk ;
bb2_g3shft1.clkf = ldclk ;
vram_data0.clkf = aluck ;
vram_data1.clkf = aluck ;
vram_data2.clkf = aluck ;
vram_data3.clkf = aluck ;
vram_data4.clkf = aluck ;
vram_data5.clkf = aluck ;
vram_data6.clkf = aluck ;
vram_data7.clkf = aluck ;
vram_data8.clkf = aluck ;
vram_data9.clkf = aluck ;
vram_data10.clkf = aluck ;
vram_data11.clkf = aluck ;
vram_data12.clkf = aluck ;
vram_data13.clkf = aluck ;
vram_data14.clkf = aluck ;
vram_data15.clkf = aluck ;
dram_data0.clkf = aluck ;
dram_data1.clkf = aluck ;
dram_data2.clkf = aluck ;
dram_data3.clkf = aluck ;
dram_data4.clkf = aluck ;
dram_data5.clkf = aluck ;
dram_data6.clkf = aluck ;
dram_data7.clkf = aluck ;
dram_data8.clkf = aluck ;
dram_data9.clkf = aluck ;
dram_data10.clkf = aluck ;
dram_data11.clkf = aluck ;
dram_data12.clkf = aluck ;
dram_data13.clkf = aluck ;
dram_data14.clkf = aluck ;
dram_data15.clkf = aluck ;
g0_func0.clkf = ldclk ;
g0_func1.clkf = ldclk ;
g0_func2.clkf = ldclk ;
g0_func3.clkf = ldclk ;
g1_func0.clkf = ldclk ;
g1_func1.clkf = ldclk ;
g1_func2.clkf = ldclk ;
g1_func3.clkf = ldclk ;
```

```
g2_func0.clkf = ldclk ;
g2_func1.clkf = ldclk ;
g2_func2.clkf = ldclk ;
g2_func3.clkf = ldclk ;
g3_func0.clkf = ldclk ;
g3_func1.clkf = ldclk ;
g3_func2.clkf = ldclk ;
g3_func3.clkf = ldclk ;
bb2_o0.setf = /sys_oe * bb2_o0 * draw_mode ;
bb2_o1.setf = /sys_oe * bb2_o0 * draw_mode ;
bb2_o2.setf = /sys_oe * bb2_o0 * draw_mode ;
bb2_o3.setf = /sys_oe * bb2_o0 * draw_mode ;
bb2_o0.rstf = /sys_oe * /bb2_o0 * draw_mode ;
bb2_o1.rstf = /sys_oe * /bb2_o0 * draw_mode ;
bb2_o2.rstf = /sys_oe * /bb2_o0 * draw_mode ;
bb2_o3.rstf = /sys_oe * /bb2_o0 * draw_mode ;
bb2_o4.setf = /sys_oe * draw_mode * pbb2_o4 ;
bb2_o5.setf = /sys_oe * draw_mode * pbb2_o4 ;
bb2_o6.setf = /sys_oe * draw_mode * pbb2_o4 ;
bb2_o7.setf = /sys_oe * draw_mode * pbb2_o4 ;
bb2_o4.rstf = /sys_oe * draw_mode * /pbb2_o4 ;
bb2_o5.rstf = /sys_oe * draw_mode * /pbb2_o4 ;
bb2_o6.rstf = /sys_oe * draw_mode * /pbb2_o4 ;
bb2_o7.rstf = /sys_oe * draw_mode * /pbb2_o4 ;
bb2_o8.setf = /sys_oe * draw_mode * pbb2_o8 ;
bb2_o9.setf = /sys_oe * draw_mode * pbb2_o8 ;
bb2_o10.setf = /sys_oe * draw_mode * pbb2_o8 ;
bb2_o11.setf = /sys_oe * draw_mode * pbb2_o8 ;
bb2_o8.rstf = /sys_oe * draw_mode * /pbb2_o8 ;
bb2_o9.rstf = /sys_oe * draw_mode * /pbb2_o8 ;
bb2_o10.rstf = /sys_oe * draw_mode * /pbb2_o8 ;
bb2_o11.rstf = /sys_oe * draw_mode * /pbb2_o8 ;
bb2_o12.setf = /sys_oe * draw_mode * pbb2_o12 ;
bb2_o13.setf = /sys_oe * draw_mode * pbb2_o12 ;
bb2_o14.setf = /sys_oe * draw_mode * pbb2_o12 ;
bb2_o15.setf = /sys_oe * draw_mode * pbb2_o12 ;
bb2_o12.rstf = /sys_oe * draw_mode * /pbb2_o12 ;
bb2_o13.rstf = /sys_oe * draw_mode * /pbb2_o12 ;
bb2_o14.rstf = /sys_oe * draw_mode * /pbb2_o12 ;
bb2_o15.rstf = /sys_oe * draw_mode * /pbb2_o12 ;
g3_func0 = /ld0sel * g3_func0
+ ld0sel * func0 ;
g3_func1 = /ld0sel * g3_func1
+ ld0sel * func1 ;
g3_func2 = /ld0sel * func0
+ ld0sel * g3_func2 ;
g3_func3 = /ld0sel * func1
+ ld0sel * g3_func3 ;
g2_func0 = /ld0sel * g2_func0
+ ld0sel * func0 ;
g2_func1 = /ld0sel * g2_func1
+ ld0sel * func1 ;
g2_func2 = /ld0sel * func0
+ ld0sel * g2_func2 ;
g2_func3 = /ld0sel * func1
+ ld0sel * g2_func3 ;
```

```
gl_func0 = /ld0sel * gl_func0
+ ld0sel * func0 ;
gl_func1 = /ld0sel * gl_func1
+ ld0sel * func1 ;
gl_func2 = /ld0sel * func0
+ ld0sel * gl_func2 ;
gl_func3 = /ld0sel * func1
+ ld0sel * gl_func3 ;
g0_func0 = /ld0sel * g0_func0
+ ld0sel * func0 ;
g0_func1 = /ld0sel * g0_func1
+ ld0sel * func1 ;
g0_func2 = /ld0sel * func0
+ ld0sel * g0_func2 ;
g0_func3 = /ld0sel * func1
+ ld0sel * g0_func3 ;
bb1_o0 = /vsel * /shift0 * /shift1 * pdram_data0
+ /vsel * shift0 * shift1 * pdram_data1
+ /vsel * /shift0 * shift1 * pdram_data2
+ /vsel * shift0 * /shift1 * pdram_data3
+ vsel * /shift0 * /shift1 * pvram_data0
+ vsel * shift0 * shift1 * pvram_data1
+ vsel * /shift0 * shift1 * pvram_data2
+ vsel * shift0 * /shift1 * pvram_data3 ;
bb1_o1 = /vsel * shift0 * /shift1 * pdram_data0
+ /vsel * /shift0 * /shift1 * pdram_data1
+ /vsel * shift0 * shift1 * pdram_data2
+ /vsel * /shift0 * shift1 * pdram_data3
+ vsel * shift0 * /shift1 * pvram_data0
+ vsel * /shift0 * /shift1 * pvram_data1
+ vsel * shift0 * shift1 * pvram_data2
+ vsel * /shift0 * shift1 * pvram_data3 ;
bb1_o2 = /vsel * shift0 * /shift1 * pdram_data1
+ /vsel * /shift0 * shift1 * pdram_data0
+ /vsel * /shift0 * /shift1 * pdram_data2
+ /vsel * shift0 * shift1 * pdram_data3
+ vsel * shift0 * /shift1 * pvram_data1
+ vsel * /shift0 * shift1 * pvram_data0
+ vsel * /shift0 * /shift1 * pvram_data2
+ vsel * shift0 * shift1 * pvram_data3 ;
bb1_o3 = /vsel * shift0 * /shift1 * pdram_data2
+ /vsel * /shift0 * shift1 * pdram_data1
+ /vsel * shift0 * shift1 * pdram_data0
+ /vsel * /shift0 * /shift1 * pdram_data3
+ vsel * shift0 * /shift1 * pvram_data2
+ vsel * /shift0 * shift1 * pvram_data1
+ vsel * shift0 * shift1 * pvram_data0
+ vsel * /shift0 * /shift1 * pvram_data3 ;
bb1_o4 = /vsel * /shift0 * /shift1 * pdram_data4
+ /vsel * shift0 * shift1 * pdram_data5
+ /vsel * /shift0 * shift1 * pdram_data6
+ /vsel * shift0 * /shift1 * pdram_data7
+ vsel * /shift0 * /shift1 * pvram_data4
+ vsel * shift0 * shift1 * pvram_data5
+ vsel * /shift0 * shift1 * pvram_data6
+ vsel * shift0 * /shift1 * pvram_data7 ;
```

```
bbl_o5 = /vsel * shift0 * /shift1 * pdram_data4
+ /vsel * /shift0 * /shift1 * pdram_data5
+ /vsel * shift0 * shift1 * pdram_data6
+ /vsel * /shift0 * shift1 * pdram_data7
+ vsel * shift0 * /shift1 * pvram_data4
+ vsel * /shift0 * /shift1 * pvram_data5
+ vsel * shift0 * shift1 * pvram_data6
+ vsel * /shift0 * shift1 * pvram_data7 ;
bbl_o6 = /vsel * shift0 * /shift1 * pdram_data5
+ /vsel * /shift0 * shift1 * pdram_data4
+ /vsel * /shift0 * /shift1 * pdram_data6
+ /vsel * shift0 * shift1 * pdram_data7
+ vsel * shift0 * /shift1 * pvram_data5
+ vsel * /shift0 * shift1 * pvram_data4
+ vsel * /shift0 * /shift1 * pvram_data6
+ vsel * shift0 * shift1 * pvram_data7 ;
bbl_o7 = /vsel * shift0 * /shift1 * pdram_data6
+ /vsel * /shift0 * shift1 * pdram_data5
+ /vsel * shift0 * shift1 * pdram_data4
+ /vsel * /shift0 * /shift1 * pdram_data7
+ vsel * shift0 * /shift1 * pvram_data6
+ vsel * /shift0 * shift1 * pvram_data5
+ vsel * shift0 * shift1 * pvram_data4
+ vsel * /shift0 * /shift1 * pvram_data7 ;
bbl_o8 = /vsel * /shift0 * /shift1 * pdram_data8
+ /vsel * shift0 * shift1 * pdram_data9
+ /vsel * /shift0 * shift1 * pdram_data10
+ /vsel * shift0 * /shift1 * pdram_data11
+ vsel * /shift0 * /shift1 * pvram_data8
+ vsel * shift0 * shift1 * pvram_data9
+ vsel * /shift0 * shift1 * pvram_data10
+ vsel * shift0 * /shift1 * pvram_data11 ;
bbl_o9 = /vsel * shift0 * /shift1 * pdram_data8
+ /vsel * /shift0 * /shift1 * pdram_data9
+ /vsel * shift0 * shift1 * pdram_data10
+ /vsel * /shift0 * shift1 * pdram_data11
+ vsel * shift0 * /shift1 * pvram_data8
+ vsel * /shift0 * /shift1 * pvram_data9
+ vsel * shift0 * shift1 * pvram_data10
+ vsel * /shift0 * shift1 * pvram_data11 ;
bbl_o10 = /vsel * shift0 * /shift1 * pdram_data9
+ /vsel * /shift0 * shift1 * pdram_data8
+ /vsel * /shift0 * /shift1 * pdram_data10
+ /vsel * shift0 * shift1 * pdram_data11
+ vsel * shift0 * /shift1 * pvram_data9
+ vsel * /shift0 * shift1 * pvram_data8
+ vsel * /shift0 * /shift1 * pvram_data10
+ vsel * shift0 * shift1 * pvram_data11 ;
bbl_o11 = /vsel * shift0 * /shift1 * pdram_data10
+ /vsel * /shift0 * shift1 * pdram_data9
+ /vsel * shift0 * shift1 * pdram_data8
+ /vsel * /shift0 * /shift1 * pdram_data11
+ vsel * shift0 * /shift1 * pvram_data10
+ vsel * /shift0 * shift1 * pvram_data9
+ vsel * shift0 * shift1 * pvram_data8
+ vsel * /shift0 * /shift1 * pvram_data11 ;
```

```
bb1_o12 = /vsel * /shift0 * /shift1 * pdram_data12
+ /vsel * shift0 * shift1 * pdram_data13
+ /vsel * /shift0 * shift1 * pdram_data14
+ /vsel * shift0 * /shift1 * pdram_data15
+ vsel * /shift0 * /shift1 * pvram_data12
+ vsel * shift0 * shift1 * pvram_data13
+ vsel * /shift0 * shift1 * pvram_data14
+ vsel * shift0 * /shift1 * pvram_data15 ;
bb1_o13 = /vsel * shift0 * /shift1 * pdram_data12
+ /vsel * /shift0 * /shift1 * pdram_data13
+ /vsel * shift0 * shift1 * pdram_data14
+ /vsel * /shift0 * shift1 * pdram_data15
+ vsel * shift0 * /shift1 * pvram_data12
+ vsel * /shift0 * /shift1 * pvram_data13
+ vsel * shift0 * shift1 * pvram_data14
+ vsel * /shift0 * shift1 * pvram_data15 ;
bb1_o14 = /vsel * shift0 * /shift1 * pdram_data13
+ /vsel * /shift0 * shift1 * pdram_data12
+ /vsel * /shift0 * /shift1 * pdram_data14
+ /vsel * shift0 * shift1 * pdram_data15
+ vsel * shift0 * /shift1 * pvram_data13
+ vsel * /shift0 * shift1 * pvram_data12
+ vsel * /shift0 * /shift1 * pvram_data14
+ vsel * shift0 * shift1 * pvram_data15 ;
bb1_o15 = /vsel * shift0 * /shift1 * pdram_data14
+ /vsel * /shift0 * shift1 * pdram_data13
+ /vsel * shift0 * shift1 * pdram_data12
+ /vsel * /shift0 * /shift1 * pdram_data15
+ vsel * shift0 * /shift1 * pvram_data14
+ vsel * /shift0 * shift1 * pvram_data13
+ vsel * shift0 * shift1 * pvram_data12
+ vsel * /shift0 * /shift1 * pvram_data15 ;
bb2_g0shft0 = /ld0sel * shift0 * /bb2_g0shft0 * /bb2_g0shft1
+ /ld0sel * /shift0 * bb2_g0shft0
+ /ld0sel * /shift0 * bb2_g0shft1
+ ld0sel * shift0 ;
bb2_g0shft1 = /ld0sel * shift1 * /bb2_g0shft0 * /bb2_g0shft1
+ /ld0sel * shift0 * /shift1 * bb2_g0shft0
+ /ld0sel * /shift0 * shift1 * bb2_g0shft0
+ /ld0sel * shift0 * /shift1 * bb2_g0shft1
+ /ld0sel * /shift0 * shift1 * bb2_g0shft1
+ ld0sel * shift1 ;
bb2_g1shft0 = /ld0sel * shift0 * /bb2_g1shft1
+ /ld0sel * /shift0 * bb2_g1shft1
+ ld0sel * shift0 ;
bb2_g1shft1 = /ld0sel * shift0 * /shift1 * bb2_g1shft1
+ /ld0sel * /shift0 * shift1 * bb2_g1shft1
+ /ld0sel * shift1 * /bb2_g1shft1
+ ld0sel * shift1 ;
bb2_g2shft0 = /ld0sel * /shift0 * bb2_g2shft0 * bb2_g2shft1
+ /ld0sel * shift0 * /bb2_g2shft0
+ /ld0sel * shift0 * /bb2_g2shft1
+ ld0sel * shift0 ;
bb2_g2shft1 = /ld0sel * shift0 * /shift1 * bb2_g2shft0 * bb2_g2shft1
+ /ld0sel * /shift0 * shift1 * bb2_g2shft0 * bb2_g2shft1
+ /ld0sel * shift1 * /bb2_g2shft0
```

```
+ /ld0sel * shift1 * /bb2_g2shft1
+ ld0sel * shift1 ;
bb2_g3shft0 = shift0 ;
bb2_g3shft1 = shift1 ;
bb2_o0 = /bb2_g0shft0 * /bb2_g0shft1 * bb1_o0
+ bb2_g0shft0 * bb2_g0shft1 * bb1_o4
+ /bb2_g0shft0 * bb2_g0shft1 * bb1_o8
+ bb2_g0shft0 * /bb2_g0shft1 * bb1_o12 ;
bb2_o1 = bb2_g0shft0 * /bb2_g0shft1 * bb1_o0
+ /bb2_g0shft0 * /bb2_g0shft1 * bb1_o4
+ bb2_g0shft0 * bb2_g0shft1 * bb1_o8
+ /bb2_g0shft0 * bb2_g0shft1 * bb1_o12 ;
bb2_o2 = bb2_g0shft0 * /bb2_g0shft1 * bb1_o4
+ /bb2_g0shft0 * bb2_g0shft1 * bb1_o0
+ /bb2_g0shft0 * /bb2_g0shft1 * bb1_o8
+ bb2_g0shft0 * bb2_g0shft1 * bb1_o12 ;
bb2_o3 = bb2_g0shft0 * /bb2_g0shft1 * bb1_o8
+ /bb2_g0shft0 * bb2_g0shft1 * bb1_o4
+ bb2_g0shft0 * bb2_g0shft1 * bb1_o0
+ /bb2_g0shft0 * /bb2_g0shft1 * bb1_o12 ;
bb2_o4 = /bb2_g1shft0 * /bb2_g1shft1 * bb1_o1
+ bb2_g1shft0 * bb2_g1shft1 * bb1_o5
+ /bb2_g1shft0 * bb2_g1shft1 * bb1_o9
+ bb2_g1shft0 * /bb2_g1shft1 * bb1_o13 ;
bb2_o5 = bb2_g1shft0 * /bb2_g1shft1 * bb1_o1
+ /bb2_g1shft0 * /bb2_g1shft1 * bb1_o5
+ bb2_g1shft0 * bb2_g1shft1 * bb1_o9
+ /bb2_g1shft0 * bb2_g1shft1 * bb1_o13 ;
bb2_o6 = bb2_g1shft0 * /bb2_g1shft1 * bb1_o5
+ /bb2_g1shft0 * bb2_g1shft1 * bb1_o1
+ /bb2_g1shft0 * /bb2_g1shft1 * bb1_o9
+ bb2_g1shft0 * bb2_g1shft1 * bb1_o13 ;
bb2_o7 = bb2_g1shft0 * /bb2_g1shft1 * bb1_o9
+ /bb2_g1shft0 * bb2_g1shft1 * bb1_o5
+ bb2_g1shft0 * bb2_g1shft1 * bb1_o1
+ /bb2_g1shft0 * /bb2_g1shft1 * bb1_o13 ;
bb2_o8 = /bb2_g2shft0 * /bb2_g2shft1 * bb1_o2
+ bb2_g2shft0 * bb2_g2shft1 * bb1_o6
+ /bb2_g2shft0 * bb2_g2shft1 * bb1_o10
+ bb2_g2shft0 * /bb2_g2shft1 * bb1_o14 ;
bb2_o9 = bb2_g2shft0 * /bb2_g2shft1 * bb1_o2
+ /bb2_g2shft0 * /bb2_g2shft1 * bb1_o6
+ bb2_g2shft0 * bb2_g2shft1 * bb1_o10
+ /bb2_g2shft0 * bb2_g2shft1 * bb1_o14 ;
bb2_o10 = bb2_g2shft0 * /bb2_g2shft1 * bb1_o6
+ /bb2_g2shft0 * bb2_g2shft1 * bb1_o2
+ /bb2_g2shft0 * /bb2_g2shft1 * bb1_o10
+ bb2_g2shft0 * bb2_g2shft1 * bb1_o14 ;
bb2_o11 = bb2_g2shft0 * /bb2_g2shft1 * bb1_o10
+ /bb2_g2shft0 * bb2_g2shft1 * bb1_o6
+ bb2_g2shft0 * bb2_g2shft1 * bb1_o2
+ /bb2_g2shft0 * /bb2_g2shft1 * bb1_o14 ;
bb2_o12 = /bb2_g3shft0 * /bb2_g3shft1 * bb1_o3
+ bb2_g3shft0 * bb2_g3shft1 * bb1_o7
+ /bb2_g3shft0 * bb2_g3shft1 * bb1_o11
+ bb2_g3shft0 * /bb2_g3shft1 * bb1_o15 ;
```

```
bb2_o13 = bb2_g3shft0 * /bb2_g3shft1 * bb1_o3
+ /bb2_g3shft0 * /bb2_g3shft1 * bb1_o7
+ bb2_g3shft0 * bb2_g3shft1 * bb1_o11
+ /bb2_g3shft0 * bb2_g3shft1 * bb1_o15 ;
bb2_o14 = bb2_g3shft0 * /bb2_g3shft1 * bb1_o7
+ /bb2_g3shft0 * bb2_g3shft1 * bb1_o3
+ /bb2_g3shft0 * /bb2_g3shft1 * bb1_o11
+ bb2_g3shft0 * bb2_g3shft1 * bb1_o15 ;
bb2_o15 = bb2_g3shft0 * /bb2_g3shft1 * bb1_o11
+ /bb2_g3shft0 * bb2_g3shft1 * bb1_o7
+ bb2_g3shft0 * bb2_g3shft1 * bb1_o3
+ /bb2_g3shft0 * /bb2_g3shft1 * bb1_o15 ;
dram_data0 = pdram_data0 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o0
+ /pdram_data0 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o0
+ /pdram_data0 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o0
+ pdram_data0 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o0
+ pdram_data0 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o0
+ /pdram_data0 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o0
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o0
+ pdram_data0 * g0_func0 * /g0_func1 * g0_func2
+ /pdram_data0 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o0 ;
dram_data1 = pdram_data1 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o4
+ /pdram_data1 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o4
+ /pdram_data1 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o4
+ pdram_data1 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o4
+ pdram_data1 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o4
+ /pdram_data1 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o4
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o4
+ pdram_data1 * g0_func0 * /g0_func1 * g0_func2
+ /pdram_data1 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o4 ;
dram_data2 = pdram_data2 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o8
+ /pdram_data2 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o8
+ /pdram_data2 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o8
+ pdram_data2 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o8
+ pdram_data2 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o8
+ /pdram_data2 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o8
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o8
+ pdram_data2 * g0_func0 * /g0_func1 * g0_func2
+ /pdram_data2 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o8 ;
dram_data3 = pdram_data3 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o12
+ /pdram_data3 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o12
+ /pdram_data3 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o12
```

```
+ pdram_data3 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o12
+ pdram_data3 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o12
+ /pdram_data3 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o12
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o12
+ pdram_data3 * g0_func0 * /g0_func1 * g0_func2
+ /pdram_data3 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o12 ;
vram_data0 = pvram_data0 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o0
+ /pvram_data0 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o0
+ /pvram_data0 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o0
+ pvram_data0 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o0
+ pvram_data0 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o0
+ /pvram_data0 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o0
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o0
+ pvram_data0 * g0_func0 * /g0_func1 * g0_func2
+ /pvram_data0 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o0 ;
vram_data1 = pvram_data1 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o4
+ /pvram_data1 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o4
+ /pvram_data1 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o4
+ pvram_data1 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o4
+ pvram_data1 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o4
+ /pvram_data1 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o4
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o4
+ pvram_data1 * g0_func0 * /g0_func1 * g0_func2
+ /pvram_data1 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o4 ;
vram_data2 = pvram_data2 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o8
+ /pvram_data2 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o8
+ /pvram_data2 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o8
+ pvram_data2 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o8
+ pvram_data2 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o8
+ /pvram_data2 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o8
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o8
+ pvram_data2 * g0_func0 * /g0_func1 * g0_func2
+ /pvram_data2 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o8 ;
vram_data3 = pvram_data3 * g0_func0 * g0_func1 * g0_func2 * /g0_func3
+ g0_func0 * g0_func1 * g0_func2 * /g0_func3 * bb2_o12
+ /pvram_data3 * g0_func0 * g0_func1 * /g0_func2 * g0_func3
+ g0_func0 * /g0_func1 * g0_func2 * g0_func3 * /bb2_o12
+ /pvram_data3 * /g0_func1 * /g0_func2 * g0_func3 * /bb2_o12
+ pvram_data3 * g0_func0 * /g0_func1 * /g0_func2 * bb2_o12
+ pvram_data3 * /g0_func0 * g0_func2 * /g0_func3 * /bb2_o12
+ /pvram_data3 * /g0_func0 * g0_func1 * /g0_func3 * bb2_o12
+ g0_func0 * g0_func1 * g0_func2 * g0_func3
```

```
+ g0_func0 * g0_func1 * /g0_func2 * bb2_o12
+ pvram_data3 * g0_func0 * /g0_func1 * g0_func2
+ /pvram_data3 * /g0_func0 * g0_func1 * g0_func3
+ /g0_func0 * g0_func2 * g0_func3 * /bb2_o12 ;
dram_data4 = pdram_data4 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o1
+ /pdram_data4 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o1
+ /pdram_data4 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o1
+ pdram_data4 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o1
+ pdram_data4 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o1
+ /pdram_data4 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o1
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o1
+ pdram_data4 * g1_func0 * /g1_func1 * g1_func2
+ /pdram_data4 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o1 ;
dram_data5 = pdram_data5 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o5
+ /pdram_data5 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o5
+ /pdram_data5 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o5
+ pdram_data5 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o5
+ pdram_data5 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o5
+ /pdram_data5 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o5
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o5
+ pdram_data5 * g1_func0 * /g1_func1 * g1_func2
+ /pdram_data5 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o5 ;
dram_data6 = pdram_data6 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o9
+ /pdram_data6 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o9
+ /pdram_data6 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o9
+ pdram_data6 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o9
+ pdram_data6 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o9
+ /pdram_data6 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o9
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o9
+ pdram_data6 * g1_func0 * /g1_func1 * g1_func2
+ /pdram_data6 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o9 ;
dram_data7 = pdram_data7 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o13
+ /pdram_data7 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o13
+ /pdram_data7 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o13
+ pdram_data7 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o13
+ pdram_data7 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o13
+ /pdram_data7 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o13
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o13
+ pdram_data7 * g1_func0 * /g1_func1 * g1_func2
+ /pdram_data7 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o13 ;
```

```
vram_data4 = pvram_data4 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o1
+ /pvram_data4 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o1
+ /pvram_data4 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o1
+ pvram_data4 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o1
+ pvram_data4 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o1
+ /pvram_data4 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o1
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o1
+ pvram_data4 * g1_func0 * /g1_func1 * g1_func2
+ /pvram_data4 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o1 ;
vram_data5 = pvram_data5 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o5
+ /pvram_data5 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o5
+ /pvram_data5 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o5
+ pvram_data5 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o5
+ pvram_data5 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o5
+ /pvram_data5 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o5
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o5
+ pvram_data5 * g1_func0 * /g1_func1 * g1_func2
+ /pvram_data5 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o5 ;
vram_data6 = pvram_data6 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o9
+ /pvram_data6 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o9
+ /pvram_data6 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o9
+ pvram_data6 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o9
+ pvram_data6 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o9
+ /pvram_data6 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o9
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o9
+ pvram_data6 * g1_func0 * /g1_func1 * g1_func2
+ /pvram_data6 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o9 ;
vram_data7 = pvram_data7 * g1_func0 * g1_func1 * g1_func2 * /g1_func3
+ g1_func0 * g1_func1 * g1_func2 * /g1_func3 * bb2_o13
+ /pvram_data7 * g1_func0 * g1_func1 * /g1_func2 * g1_func3
+ g1_func0 * /g1_func1 * g1_func2 * g1_func3 * /bb2_o13
+ /pvram_data7 * /g1_func1 * /g1_func2 * g1_func3 * /bb2_o13
+ pvram_data7 * g1_func0 * /g1_func1 * /g1_func2 * bb2_o13
+ pvram_data7 * /g1_func0 * g1_func2 * /g1_func3 * /bb2_o13
+ /pvram_data7 * /g1_func0 * g1_func1 * /g1_func3 * bb2_o13
+ g1_func0 * g1_func1 * g1_func2 * g1_func3
+ g1_func0 * g1_func1 * /g1_func2 * bb2_o13
+ pvram_data7 * g1_func0 * /g1_func1 * g1_func2
+ /pvram_data7 * /g1_func0 * g1_func1 * g1_func3
+ /g1_func0 * g1_func2 * g1_func3 * /bb2_o13 ;
dram_data8 = pdram_data8 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o2
+ /pdram_data8 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o2
```

```
+ /pdram_data8 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o2
+ pdram_data8 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o2
+ pdram_data8 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o2
+ /pdram_data8 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o2
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o2
+ pdram_data8 * g2_func0 * /g2_func1 * g2_func2
+ /pdram_data8 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o2 ;
dram_data9 = pdram_data9 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o6
+ /pdram_data9 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o6
+ /pdram_data9 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o6
+ pdram_data9 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o6
+ pdram_data9 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o6
+ /pdram_data9 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o6
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o6
+ pdram_data9 * g2_func0 * /g2_func1 * g2_func2
+ /pdram_data9 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o6 ;
dram_data10 = pdram_data10 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o10
+ /pdram_data10 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o10
+ /pdram_data10 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o10
+ pdram_data10 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o10
+ pdram_data10 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o10
+ /pdram_data10 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o10
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o10
+ pdram_data10 * g2_func0 * /g2_func1 * g2_func2
+ /pdram_data10 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o10 ;
dram_data11 = pdram_data11 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o14
+ /pdram_data11 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o14
+ /pdram_data11 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o14
+ pdram_data11 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o14
+ pdram_data11 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o14
+ /pdram_data11 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o14
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o14
+ pdram_data11 * g2_func0 * /g2_func1 * g2_func2
+ /pdram_data11 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o14 ;
vram_data8 = pvram_data8 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o2
+ /pvram_data8 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o2
+ /pvram_data8 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o2
+ pvram_data8 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o2
+ pvram_data8 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o2
+ /pvram_data8 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o2
```

```
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o2
+ pvram_data8 * g2_func0 * /g2_func1 * g2_func2
+ /pvram_data8 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o2 ;
vram_data9 = pvram_data9 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o6
+ /pvram_data9 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o6
+ /pvram_data9 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o6
+ pvram_data9 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o6
+ pvram_data9 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o6
+ /pvram_data9 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o6
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o6
+ pvram_data9 * g2_func0 * /g2_func1 * g2_func2
+ /pvram_data9 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o6 ;
vram_data10 = pvram_data10 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o10
+ /pvram_data10 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o10
+ /pvram_data10 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o10
+ pvram_data10 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o10
+ pvram_data10 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o10
+ /pvram_data10 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o10
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o10
+ pvram_data10 * g2_func0 * /g2_func1 * g2_func2
+ /pvram_data10 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o10 ;
vram_data11 = pvram_data11 * g2_func0 * g2_func1 * g2_func2 * /g2_func3
+ g2_func0 * g2_func1 * g2_func2 * /g2_func3 * bb2_o14
+ /pvram_data11 * g2_func0 * g2_func1 * /g2_func2 * g2_func3
+ g2_func0 * /g2_func1 * g2_func2 * g2_func3 * /bb2_o14
+ /pvram_data11 * /g2_func1 * /g2_func2 * g2_func3 * /bb2_o14
+ pvram_data11 * g2_func0 * /g2_func1 * /g2_func2 * bb2_o14
+ pvram_data11 * /g2_func0 * g2_func2 * /g2_func3 * /bb2_o14
+ /pvram_data11 * /g2_func0 * g2_func1 * /g2_func3 * bb2_o14
+ g2_func0 * g2_func1 * g2_func2 * g2_func3
+ g2_func0 * g2_func1 * /g2_func2 * bb2_o14
+ pvram_data11 * g2_func0 * /g2_func1 * g2_func2
+ /pvram_data11 * /g2_func0 * g2_func1 * g2_func3
+ /g2_func0 * g2_func2 * g2_func3 * /bb2_o14 ;
dram_data12 = pdram_data12 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o3
+ /pdram_data12 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o3
+ /pdram_data12 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o3
+ pdram_data12 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o3
+ pdram_data12 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o3
+ /pdram_data12 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o3
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o3
+ pdram_data12 * g3_func0 * /g3_func1 * g3_func2
+ /pdram_data12 * /g3_func0 * g3_func1 * g3_func3
```

```
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o3 ;
dram_data13 = pdram_data13 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o7
+ /pdram_data13 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o7
+ /pdram_data13 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o7
+ pdram_data13 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o7
+ pdram_data13 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o7
+ /pdram_data13 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o7
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o7
+ pdram_data13 * g3_func0 * /g3_func1 * g3_func2
+ /pdram_data13 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o7 ;
dram_data14 = pdram_data14 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o11
+ /pdram_data14 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o11
+ /pdram_data14 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o11
+ pdram_data14 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o11
+ pdram_data14 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o11
+ /pdram_data14 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o11
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o11
+ pdram_data14 * g3_func0 * /g3_func1 * g3_func2
+ /pdram_data14 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o11 ;
dram_data15 = pdram_data15 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o15
+ /pdram_data15 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o15
+ /pdram_data15 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o15
+ pdram_data15 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o15
+ pdram_data15 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o15
+ /pdram_data15 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o15
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o15
+ pdram_data15 * g3_func0 * /g3_func1 * g3_func2
+ /pdram_data15 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o15 ;
vram_data12 = pvram_data12 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o3
+ /pvram_data12 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o3
+ /pvram_data12 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o3
+ pvram_data12 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o3
+ pvram_data12 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o3
+ /pvram_data12 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o3
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o3
+ pvram_data12 * g3_func0 * /g3_func1 * g3_func2
+ /pvram_data12 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o3 ;
vram_data13 = pvram_data13 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o7
+ /pvram_data13 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
```

```
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o7
+ /pvram_data13 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o7
+ pvram_data13 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o7
+ pvram_data13 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o7
+ /pvram_data13 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o7
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o7
+ pvram_data13 * g3_func0 * /g3_func1 * g3_func2
+ /pvram_data13 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o7 ;
vram_data14 = pvram_data14 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o11
+ /pvram_data14 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o11
+ /pvram_data14 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o11
+ pvram_data14 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o11
+ pvram_data14 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o11
+ /pvram_data14 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o11
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o11
+ pvram_data14 * g3_func0 * /g3_func1 * g3_func2
+ /pvram_data14 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o11 ;
vram_data15 = pvram_data15 * g3_func0 * g3_func1 * g3_func2 * /g3_func3
+ g3_func0 * g3_func1 * g3_func2 * /g3_func3 * bb2_o15
+ /pvram_data15 * g3_func0 * g3_func1 * /g3_func2 * g3_func3
+ g3_func0 * /g3_func1 * g3_func2 * g3_func3 * /bb2_o15
+ /pvram_data15 * /g3_func1 * /g3_func2 * g3_func3 * /bb2_o15
+ pvram_data15 * g3_func0 * /g3_func1 * /g3_func2 * bb2_o15
+ pvram_data15 * /g3_func0 * g3_func2 * /g3_func3 * /bb2_o15
+ /pvram_data15 * /g3_func0 * g3_func1 * /g3_func3 * bb2_o15
+ g3_func0 * g3_func1 * g3_func2 * g3_func3
+ g3_func0 * g3_func1 * /g3_func2 * bb2_o15
+ pvram_data15 * g3_func0 * /g3_func1 * g3_func2
+ /pvram_data15 * /g3_func0 * g3_func1 * g3_func3
+ /g3_func0 * g3_func2 * g3_func3 * /bb2_o15 ;
; ***************These equations must be added at the end of the
;                the equations after each Tango conversion
pbb1_o0 = {bb1_o0 } ;
pbb1_o1 = {bb1_o1 } ;
pbb1_o2 = {bb1_o2 } ;
pbb1_o3 = {bb1_o3 } ;
pbb1_o4 = {bb1_o4 } ;
pbb1_o5 = {bb1_o5 } ;
pbb1_o6 = {bb1_o6 } ;
pbb1_o7 = {bb1_o7 } ;
pbb1_o8 = {bb1_o8 } ;
pbb1_o9 = {bb1_o9 } ;
pbb1_o10 = {bb1_o10 } ;
pbb1_o11 = {bb1_o11 } ;
pbb1_o12 = {bb1_o12 } ;
pbb1_o13 = {bb1_o13 } ;
pbb1_o14 = {bb1_o14 } ;
pbb1_o15 = {bb1_o15 } ;
pvram_data0 = {vram_data0} ;
pvram_data1 = {vram_data1} ;
```

```
pvram_data2 = {vram_data2} ;
pvram_data3 = {vram_data3} ;
pvram_data4 = {vram_data4} ;
pvram_data5 = {vram_data5} ;
pvram_data6 = {vram_data6} ;
pvram_data7 = {vram_data7} ;
pvram_data8 = {vram_data8} ;
pvram_data9 = {vram_data9} ;
pvram_data10 = {vram_data10} ;
pvram_data11 = {vram_data11} ;
pvram_data12 = {vram_data12} ;
pvram_data13 = {vram_data13} ;
pvram_data14 = {vram_data14} ;
pvram_data15 = {vram_data15} ;
pdram_data0 = {dram_data0} ;
pdram_data1 = {dram_data1} ;
pdram_data2 = {dram_data2} ;
pdram_data3 = {dram_data3} ;
pdram_data4 = {dram_data4} ;
pdram_data5 = {dram_data5} ;
pdram_data6 = {dram_data6} ;
pdram_data7 = {dram_data7} ;
pdram_data8 = {dram_data8} ;
pdram_data9 = {dram_data9} ;
pdram_data10 = {dram_data10} ;
pdram_data11 = {dram_data11} ;
pdram_data12 = {dram_data12} ;
pdram_data13 = {dram_data13} ;
pdram_data14 = {dram_data14} ;
pdram_data15 = {dram_data15} ;
pbb2_o0 = {bb2_o0} ;
pbb2_o4 = {bb2_o4} ;
pbb2_o8 = {bb2_o8} ;
pbb2_o12 = {bb2_o12} ;
;------------------------------- Simulation Segment ------------
SIMULATION
;-------------------------------------------------------------------
```

FITTER FILE

Note: This file has been condensed in order to save trees.

```
PALASM 4 v1.4 MACH FITR - MARKET RELEASE (1-29-92)
(C) - COPYRIGHT ADVANCED MICRO DEVICES INC., 1990,91,92
*********************************************
Mach PLD Fitter - v 1.62          PP
*********************************************
*** Timing Analysis for Signals
Parameter   Min   Max          Signal List (Those having Max delay.)
     Tsu     1     1          PBB1_O3        BB2_G3SHFT1        PBB1_O2
                              PBB1_O1           PBB1_O0     BB2_G2SHFT1
                              PBB1_O7        PDRAM_DATA0
     Tco     0     0          PBB2_O4           PBB2_O8         PBB2_O12
                          PDRAM_DATA0        PDRAM_DATA3     PDRAM_DATA1
                          PVRAM_DATA0           G3_FUNC2
     Tcr     1     1           BB2_O5            BB2_O7           BB2_O6
                               BB2_O4            BB2_O11          BB2_O9
                               BB2_O8        PVRAM_DATA12

.
Key:
Tpd - Combinatorial propagation delay, input to output
Tsu - Combinatorial setup delay before clock
Tco - clock to output (register output to combinatorial output)
Tcr - Clock to register setup delay
.......(Register output thru combinatorial logic to reg input)
All delay values are expressed in terms of array passes
*** Device Resource Checks
                Available          Used          Remaining
     Clocks:        4                4               0
       Pins:       70               68               2      ->     97%
  I/O Macro:       64               52              12
Total Macro:      128              .88              40
Product Terms:    512              379              88      ->     82%
MACH-PLD Resource Checks OK!
Partitioning Design into Blocks...
|> INFORMATION F050 - Device Utilization.......  *: 87 %
```

```
*** Feedback Map - PP
Gbl Inp .--.      I/O    .--+--A--+--.  I/O              I/O   .--+--B--+--.  I/O
     | 0|  PDRAM_DA  0|      |25| PBB1_O1      PDRAM_DA  0|       |25| PBB1_O9
     | 1|  PDRAM_DA  1|      |24| SHIFT0       PBB1_O13  1|       |24|
     | 2|  PDRAM_DA  2|      |23| SHIFT1       PDRAM_DA  2|       |23| PBB1_O5
     | 3|  PBB1_O0 : 3|      |22| PBB1_O4      SYS_ACCE  3|       |22| LD0SEL
     | 4|  BB2_G0SH  4|      |21| BB2_G0S      PBB1_O1 : 4|       |21| BB2_G1S
     | 5|           | 5|     |20| SYS_ACC      PVRAM_DA  5|       |20| DRAW_MO
     '--'  PVRAM_DA  6|      |19| PBB2_O0       SHIFT1 : 6|       |19|
           PVRAM_DA  7|      |18| LD0SEL       PDRAM_DA  7|       |18| PBB2_O4
           PVRAM_DA  8|      |17| PBB1_O8      PVRAM_DA  8|       |17|
            VSEL : 9|        |16| DRAW_MO        VSEL : 9|        |16| SHIFT0
                   |10|      |15: WE_OE        PDRAM_DA 10|       |15: WE_OE
            SYS_OE :11|      |14|               SYS_OE :11|       |14: PVRAM_D
           PVRAM_DA 12|      |13: PDRAM_D      PVRAM_DA 12|       |13: BB2_G1S
                     '--+--C--+--'                       '--+--D--+--'
           PDRAM_DA  0|      |25|              PBB1_O3 : 0|       |25| WE_OE
           PBB2_O8 : 1|      |24| SHIFT0       PDRAM_DA  1|       |24| SHIFT0
           PVRAM_DA  2|      |23| BB2_G2S      PBB1_O11  2|       |23| SHIFT1
           PVRAM_DA  3|      |22| LD0SEL       SYS_ACCE  3|       |22|
           PDRAM_DA  4|      |21| SYS_ACC      PVRAM_DA  4|       |21| BB2_G3S
           PDRAM_DA  5|      |20| SYS_OE       PDRAM_DA  5|       |20| SYS_OE
           PDRAM_DA  6|      |19| PBB1_O2      PBB2_O12  6|       |19| PDRAM_D
           PVRAM_DA  7|      |18| VSEL         PBB1_O15  7|       |18|
           DRAW_MOD  8|      |17|              PVRAM_DA  8|       |17| DRAW_MO
           PBB1_O6 : 9|      |16| WE_OE          VSEL : 9|        |16| PVRAM_D
                   |10|      |15: SHIFT1       PBB1_O7 :10|       |15: PVRAM_D
           PVRAM_DA 11|      |14: PBB1_O1      BB2_G3SH 11|       |14|
           BB2_G2SH 12|      |13: PBB1_O1      PDRAM_DA 12|       |13|
                     '--+--E--+--'                       '--+--F--+--'
           G0_FUNC1  0|      |25|               VALUOE : 0|       |25| PVRAM_D
           PDRAM_DA  1|      |24| BB2_O12      G1_FUNC2  1|       |24|
           PDRAM_DA  2|      |23|              PVRAM_DA  2|       |23|
            FUNC1 : 3|       |22| LD0SEL       PVRAM_DA  3|       |22| LD0SEL
            BB2_O4 : 4|      |21| PVRAM_D      PDRAM_DA  4|       |21| BB2_O13
           PDRAM_DA  5|      |20| BB2_O8       PDRAM_DA  5|       |20| G1_FUNC
           PVRAM_DA  6|      |19| PBB2_O0      PDRAM_DA  6|       |19|
           PVRAM_DA  7|      |18| FUNC0        G1_FUNC0  7|       |18| FUNC0
           PVRAM_DA  8|      |17|              PDRAM_DA  8|       |17|
            VALUOE : 9|      |16| G0_FUNC      G1_FUNC1  9|       |16| BB2_O1
           G0_FUNC2 10|      |15|               BB2_O5 :10|       |15|
           G0_FUNC3 11|      |14|              PVRAM_DA 11|       |14: DALUOE
           PDRAM_DA 12|      |13: DALUOE         FUNC1 :12|       |13: BB2_O9
                     '--+--G--+--'                       '--+--H--+--'
            VALUOE : 0|      |25|              PDRAM_DA  0|       |25|
           PDRAM_DA  1|      |24| PVRAM_D      PDRAM_DA  1|       |24|
           G2_FUNC3  2|      |23| PDRAM_D      PDRAM_DA  2|       |23|
            DALUOE : 3|      |22| LD0SEL        DALUOE : 3|       |22|
                   | 4|      |21|              PDRAM_DA  4|       |21| PVRAM_D
           PVRAM_DA  5|      |20| BB2_O6         BB2_O3 : 5|      |20| BB2_O11
           PVRAM_DA  6|      |19| FUNC0        PVRAM_DA  6|       |19| FUNC0
           PDRAM_DA  7|      |18| G2_FUNC      PVRAM_DA  7|       |18| LD0SEL
           PVRAM_DA  8|      |17| BB2_O2       PVRAM_DA  8|       |17| VALUOE
           PDRAM_DA  9|      |16| G2_FUNC      G3_FUNC0  9|       |16| G3_FUNC
           G2_FUNC2 10|      |15|              G3_FUNC1 10|       |15: BB2_O15
           BB2_O10 :11|      |14: BB2_O14      G3_FUNC2 11|       |14|
             FUNC1 :12|      |13|                FUNC1 :12|       |13: BB2_O7
                     '--+-u--u+--'                       '--+-u--u+--'
```

```
*** Logic Map - PP
Gbl Inp .--.      I/O   .--+--A--+--.  I/O            I/O   .--+--B--+--.  I/O
 BB2_CLK| 0|  PBB2_O0 | 0| 4    |25|               PBB2_O4 | 0| 4    |25|
  BB_CLK| 1|          | 1| *    |24|                       | 1| *    |24|
  LD0SEL| 2|  PBB1_O12  2| 8    |23|               PBB1_O8   2| 8    |23|
   ALUCK| 3|          | 3| *    |22|                       | 3| *    |22|
   LDCLK| 4|  PBB1_O14  4| 8    |21|               PBB1_O10  4| 8    |21|
  DALUOE| 5|          | 5| *    |20|                       | 5| *    |20|
        '--'  PBB1_O13  6| 8    |19|               PBB1_O9 | 6| 8    |19|
                      | 7| *    |18|                BB2_O6 | 7| 4    |18|
              PBB1_O15  8| 8    |17|               PBB1_O11  8| 8    |17|
              BB2_G0SH  9| 5    |16|               BB2_G1SH  9| 3    |16|
                BB2_O1 |10| 4   .|15|                BB2_O7 |10| 4   .|15|
              BB2_G0SH 11| 4   *|14|               BB2_G1SH 11| 4   .|14|
                BB2_O2 |12| 4   4|13| BB2_O3         BB2_O5 |12| 4   *|13|
                      '--+--C--+--'                        '--+--D--+--'
               PBB1_O4 | 0| 8    |25|               PBB2_O12  0| 4    |25|
                      | 1| *    |24|                       | 1| *    |24|
               PBB1_O5 | 2| 8    |23|               PBB1_O0 | 2| 8    |23|
                      | 3| *    |22|                       | 3| *    |22|
               PBB1_O6 | 4| 8    |21|               PBB1_O1 | 4| 8    |21|
                      | 5| *    |20|                       | 5| *    |20|
               PBB1_O7 | 6| 8    |19|               PBB1_O2 | 6| 8    |19|
                      | 7| *    |18|               BB2_G3SH  7| 1    |18|
               BB2_O10 | 8| 4    |17|               PBB1_O3 | 8| 8    |17|
              BB2_G2SH  9| 5    |16|               BB2_G3SH  9| 1    |16|
               PBB2_O8 |10| 4   .|15|               BB2_O13 |10| 4   .|15|
                BB2_O9 |11| 4   *|14|                       |11| *   4|14| BB2_O15
               BB2_O11 |12| 4   4|13| BB2_G2S       BB2_O14 |12| 4   .|13|
                      '--+--E--+--'                        '--+--F--+--'
              PDRAM_DA  0| 4    |25|               PVRAM_DA  0| 4    |25|
              G0_FUNC0  1| 2    |24|               G1_FUNC0  1| 2    |24|
              PDRAM_DA  2| 4    |23|               PVRAM_DA  2| 4    |23|
              G0_FUNC1  3| 2    |22|               G1_FUNC1  3| 2    |22|
              PDRAM_DA  4| 4    |21|               PDRAM_DA  4| 4    |21|
              G0_FUNC2  5| 2    |20|               G1_FUNC2  5| 2    |20|
              PVRAM_DA  6| 4    |19|               PDRAM_DA  6| 4    |19|
              G0_FUNC3  7| 2    |18|               G1_FUNC3  7| 2    |18|
              PDRAM_DA  8| 4    |17|               PDRAM_DA  8| 4    |17|
                      | 9| .    |16|                       | 9| .    |16|
              PVRAM_DA 10| 4   .|15|               PVRAM_DA 10| 4   .|15|
                      |11| .   4|14| PVRAM_D               |11| .   4|14| PVRAM_D
              PVRAM_DA 12| 4   .|13|               PDRAM_DA 12| 4   .|13|
                      '--+--G--+--'                        '--+--H--+--'
              PVRAM_DA  0| 4    |25|               PVRAM_DA  0| 4    |25|
              G2_FUNC0  1| 2    |24|               G3_FUNC3  1| 2    |24|
              PVRAM_DA  2| 4    |23|               PVRAM_DA  2| 4    |23|
              G2_FUNC1  3| 2    |22|               G3_FUNC0  3| 2    |22|
              PDRAM_DA  4| 4    |21|               PVRAM_DA  4| 4    |21|
              G2_FUNC2  5| 2    |20|               G3_FUNC1  5| 2    |20|
              PVRAM_DA  6| 4    |19|               PDRAM_DA  6| 4    |19|
              G2_FUNC3  7| 2    |18|               G3_FUNC2  7| 2    |18|
              PDRAM_DA  8| 4    |17|               PDRAM_DA  8| 4    |17|
                      | 9| .    |16|                       | 9|      |16|
             PDRAM_DA 10| 4   .|15|                PDRAM_DA 10| 4   .|15|
                      |11| .   4|14| PVRAM_D               |11| .   4|14| PDRAM_D
             PDRAM_DA 12| 4   .|13|                PVRAM_DA 12| 4   .|13|
                      '--+-u--u+--'                        '--+-u--u+--'
```

```
*** Pin Map - PP
                                                    DALUOE
                          PBB2_O0                   | PVRAM_DATA12
                       PBB1_O12 |                   | | PVRAM_DATA13
                     PBB1_O14 | |                   | | | PVRAM_DATA14
                   PBB1_O13 | | |                   | | | | PDRAM_DATA12
                 PBB1_O15 | | | |                   | | | | | PDRAM_DATA14
                 SHIFT1 | | | | |                   | | | | | | PDRAM_DATA15
               SHIFT0 | | | | | |                   | | | | | | | PVRAM_DATA15
                   . | | | | | | |                  | | | | | | | | PDRAM_DATA13
                   | | | | | | | |                  | | | | | | | | |
             .------'-'-'-'-'-'-'-'---o---'-'-'-'-'-'-'-'-'----.
             |    1 1                  8 8 8 8 8 7 7 7 7 7     |
             |    1 0 9 8 7 6 5 4 3 2 1 4 3 2 1 0 9 8 7 6 5     |
         VSEL|12                                         Gnd 74|
    DRAW_MODE|13  G          V G V                           73|PVRAM_DATA11
        WE_OE|14  n          c n c                           72|PDRAM_DATA9
    PBB1_O11|15   d          c d c                           71|PDRAM_DATA11
      PBB1_O9|16                                             70|PDRAM_DATA10
     PBB1_O10|17                                             69|PVRAM_DATA10
      PBB1_O8|18                                             68|PDRAM_DATA8
      PBB2_O4|19                                             67|PVRAM_DATA9
      BB2_CLK|20                                             66|PVRAM_DATA8
             |21 Vcc           MACH-230                      65|LDCLK
             |22 Gnd                                     Gnd 64|
       BB_CLK|23                                         Vcc 63|
      PBB1_O4|24                                             62|ALUCK
      PBB1_O5|25                                             61|PVRAM_DATA6
      PBB1_O6|26                                             60|PVRAM_DATA7
      PBB1_O7|27                                             59|PDRAM_DATA4
      VALUOE|28                                              58|PDRAM_DATA6
      PBB2_O8|29             V G V                       G   57|PDRAM_DATA5
       SYS_OE|30             c n c                       n   56|PVRAM_DATA4
   SYS_ACCESS|31             c d c                       d   55|PDRAM_DATA7
             |32 Gnd                                         54|PVRAM_DATA5
             |    3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 5 5 5 5    |
             |    3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3    |
             '------.-.-.-.-.-.-.-.--------.-.-.-.-.-.-.-.-----'
                    | | | | | | | |        | | | | | | | |
                    ' | | | | | | |        | | | | | | | |
              FUNC1 | | | | | | |          | | | | | | | PVRAM_DATA3
                FUNC0 | | | | | |          | | | | | | PVRAM_DATA2
                 PBB1_O3 | | | |           | | | | | PVRAM_DATA1
                   PBB1_O2 | | |           | | | | PDRAM_DATA2
                     PBB1_O1 | |           | | | PVRAM_DATA0
                       PBB1_O0 |           | | PDRAM_DATA1
                        PBB2_O12 |         | PDRAM_DATA3
                             LD0SEL      PDRAM_DATA0
The Design Doc is  stored in ===> PP_FINAL.Rpt
The Jedec Data is  stored in ===> PP_FINAL.Jed
The Placements are stored in ===> PP_FINAL.Plc
%% FITR %% Error Count: 0, Warning Count: 48
%% FITR %% File Processed Successfully. - File: PP_FINAL.PDC
```

PLACEMENT FILE

```
Flags Used:            Unplace=False          Max Packing=True
Flags Used:         Expand Small=False         Expand All=False
; FITR generated placements
Node   2          BB2_O0     Reg          ; A  0
Node   4         BB1_O12     Reg          ; A  2
Node   6         BB1_O14     Reg          ; A  4
Node   8         BB1_O13     Reg          ; A  6
Node  10         BB1_O15     Reg          ; A  8
Pin    8          SHIFT1   ; Inp          ; A 10
Pin    9          SHIFT0   ; Inp          ; A 12
Pin   12            VSEL   ; Inp          ; B 14
Pin   13       DRAW_MODE   ; Inp          ; B 12
Pin   14           WE_OE   ; Inp          ; B 10
Node  26         BB1_O11     Reg          ; B  8
Node  24          BB1_O9     Reg          ; B  6
Node  22         BB1_O10     Reg          ; B  4
Node  20          BB1_O8     Reg          ; B  2
Pin   19         PBB2_O4     Reg          ; B  0
Pin   20         BB2_CLK   ; Inp          ; I  0
Pin   23          BB_CLK   ; Inp          ; I  1
Node  34          BB1_O4     Reg          ; C  0
Node  36          BB1_O5     Reg          ; C  2
Node  38          BB1_O6     Reg          ; C  4
Node  40          BB1_O7     Reg          ; C  6
Pin   28          VALUOE   ; Inp          ; C  8
Pin   29         PBB2_O8     Reg          ; C 10
Pin   30          SYS_OE   ; Inp          ; C 12
Pin   31      SYS_ACCESS   ; Inp          ; C 14
Pin   34           FUNC1   ; Inp          ; D 12
Pin   35           FUNC0   ; Inp          ; D 10
Node  58          BB1_O3     Reg          ; D  8
Node  56          BB1_O2     Reg          ; D  6
Node  54          BB1_O1     Reg          ; D  4
Node  52          BB1_O0     Reg          ; D  2
Pin   40        PBB2_O12     Reg          ; D  0
Pin   41          LD0SEL   ; Inp          ; I  2
Pin   45     PDRAM_DATA0     Reg          ; E  0
Pin   46     PDRAM_DATA3     Reg          ; E  2
Pin   47     PDRAM_DATA1     Reg          ; E  4
Pin   48     PVRAM_DATA0     Reg          ; E  6
Pin   49     PDRAM_DATA2     Reg          ; E  8
Pin   50     PVRAM_DATA1     Reg          ; E 10
Pin   51     PVRAM_DATA2     Reg          ; E 12
Pin   52     PVRAM_DATA3     Reg          ; E 14
Pin   54     PVRAM_DATA5     Reg          ; F 14
Pin   55     PDRAM_DATA7     Reg          ; F 12
Pin   56     PVRAM_DATA4     Reg          ; F 10
Pin   57     PDRAM_DATA5     Reg          ; F  8
Pin   58     PDRAM_DATA6     Reg          ; F  6
Pin   59     PDRAM_DATA4     Reg          ; F  4
Pin   60     PVRAM_DATA7     Reg          ; F  2
Pin   61     PVRAM_DATA6     Reg          ; F  0
Pin   62           ALUCK   ; Inp          ; I  3
Pin   65           LDCLK   ; Inp          ; I  4
Pin   66     PVRAM_DATA8     Reg          ; G  0
```

Pin	67	PVRAM_DATA9	Reg	; G 2
Pin	68	PDRAM_DATA8	Reg	; G 4
Pin	69	PVRAM_DATA10	Reg	; G 6
Pin	70	PDRAM_DATA10	Reg	; G 8
Pin	71	PDRAM_DATA11	Reg	; G 10
Pin	72	PDRAM_DATA9	Reg	; G 12
Pin	73	PVRAM_DATA11	Reg	; G 14
Pin	75	PDRAM_DATA13	Reg	; H 14
Pin	76	PVRAM_DATA15	Reg	; H 12
Pin	77	PDRAM_DATA15	Reg	; H 10
Pin	78	PDRAM_DATA14	Reg	; H 8
Pin	79	PDRAM_DATA12	Reg	; H 6
Pin	80	PVRAM_DATA14	Reg	; H 4
Pin	81	PVRAM_DATA13	Reg	; H 2
Pin	82	PVRAM_DATA12	Reg	; H 0
Pin	83	DALUOE	; Inp	; I 5
Pin	3	PBB2_O0	Reg	; A 0
Pin	4	PBB1_O12	Reg	; A 2
Pin	5	PBB1_O14	Reg	; A 4
Pin	6	PBB1_O13	Reg	; A 6
Pin	7	PBB1_O15	Reg	; A 8
Node	11	BB2_G0SHFT1	Reg	; A 9
Node	12	BB2_O1	Reg	; A 10
Node	13	BB2_G0SHFT0	Reg	; A 11
Node	14	BB2_O2	Reg	; A 12
Node	15	BB2_O3	Reg	; A 13
Node	18	BB2_O4	Reg	; B 0
Pin	18	PBB1_O8	Reg	; B 2
Pin	17	PBB1_O10	Reg	; B 4
Pin	16	PBB1_O9	Reg	; B 6
Node	25	BB2_O6	Reg	; B 7
Pin	15	PBB1_O11	Reg	; B 8
Node	27	BB2_G1SHFT0	Reg	; B 9
Node	28	BB2_O7	Reg	; B 10
Node	29	BB2_G1SHFT1	Reg	; B 11
Node	30	BB2_O5	Reg	; B 12
Pin	24	PBB1_O4	Reg	; C 0
Pin	25	PBB1_O5	Reg	; C 2
Pin	26	PBB1_O6	Reg	; C 4
Pin	27	PBB1_O7	Reg	; C 6
Node	42	BB2_O10	Reg	; C 8
Node	43	BB2_G2SHFT1	Reg	; C 9
Node	44	BB2_O8	Reg	; C 10
Node	45	BB2_O9	Reg	; C 11
Node	46	BB2_O11	Reg	; C 12
Node	47	BB2_G2SHFT0	Reg	; C 13
Node	50	BB2_O12	Reg	; D 0
Pin	39	PBB1_O0	Reg	; D 2
Pin	38	PBB1_O1	Reg	; D 4
Pin	37	PBB1_O2	Reg	; D 6
Node	57	BB2_G3SHFT1	Reg	; D 7
Pin	36	PBB1_O3	Reg	; D 8
Node	59	BB2_G3SHFT0	Reg	; D 9
Node	60	BB2_O13	Reg	; D 10
Node	62	BB2_O14	Reg	; D 12
Node	64	BB2_O15	Reg	; D 14

```
Node  66      DRAM_DATA0   Reg         ; E  0
Node  67        G0_FUNC0   Reg         ; E  1
Node  68      DRAM_DATA3   Reg         ; E  2
Node  69        G0_FUNC1   Reg         ; E  3
Node  70      DRAM_DATA1   Reg         ; E  4
Node  71        G0_FUNC2   Reg         ; E  5
Node  72      VRAM_DATA0   Reg         ; E  6
Node  73        G0_FUNC3   Reg         ; E  7
Node  74      DRAM_DATA2   Reg         ; E  8
Node  76      VRAM_DATA1   Reg         ; E 10
Node  78      VRAM_DATA2   Reg         ; E 12
Node  80      VRAM_DATA3   Reg         ; E 14
Node  82      VRAM_DATA6   Reg         ; F  0
Node  83        G1_FUNC0   Reg         ; F  1
Node  84      VRAM_DATA7   Reg         ; F  2
Node  85        G1_FUNC1   Reg         ; F  3
Node  86      DRAM_DATA4   Reg         ; F  4
Node  87        G1_FUNC2   Reg         ; F  5
Node  88      DRAM_DATA6   Reg         ; F  6
Node  89        G1_FUNC3   Reg         ; F  7
Node  90      DRAM_DATA5   Reg         ; F  8
Node  92      VRAM_DATA4   Reg         ; F 10
Node  94      DRAM_DATA7   Reg         ; F 12
Node  96      VRAM_DATA5   Reg         ; F 14
Node  98      VRAM_DATA8   Reg         ; G  0
Node  99        G2_FUNC0   Reg         ; G  1
Node 100      VRAM_DATA9   Reg         ; G  2
Node 101        G2_FUNC1   Reg         ; G  3
Node 102      DRAM_DATA8   Reg         ; G  4
Node 103        G2_FUNC2   Reg         ; G  5
Node 104     VRAM_DATA10   Reg         ; G  6
Node 105        G2_FUNC3   Reg         ; G  7
Node 106     DRAM_DATA10   Reg         ; G  8
Node 108     DRAM_DATA11   Reg         ; G 10
Node 110      DRAM_DATA9   Reg         ; G 12
Node 112     VRAM_DATA11   Reg         ; G 14
Node 114     VRAM_DATA12   Reg         ; H  0
Node 115        G3_FUNC3   Reg         ; H  1
Node 116     VRAM_DATA13   Reg         ; H  2
Node 117        G3_FUNC0   Reg         ; H  3
Node 118     VRAM_DATA14   Reg         ; H  4
Node 119        G3_FUNC1   Reg         ; H  5
Node 120     DRAM_DATA12   Reg         ; H  6
Node 121        G3_FUNC2   Reg         ; H  7
Node 122     DRAM_DATA14   Reg         ; H  8
Node 124     DRAM_DATA15   Reg         ; H 10
Node 126     VRAM_DATA15   Reg         ; H 12
Node 128     DRAM_DATA13   Reg         ; H 14
;
; Unused Pins & Nodes
;
; -> None Found.
;
```

NOTES